To Recommend or not Recommend, That's The GraphBandit Question

Colby Ziyu Wang & Tejas Vyas

December 2024

1 Introduction

The focus of this paper is the problem of recipe recommendation. Given a user and a recipe, the goal is to determine whether the recipe should be recommended (output 1) or not recommended (output 0). Each user and recipe is represented by a learned embedding, derived from historical interaction data and recipe attributes. These embeddings form the basis for making accurate recommendation decisions.

2 Methodology

In this section, we provide a detailed description of the proposed model, which integrates a Graph Neural Network (GNN) for learning embeddings and a Contextual Bandit for making recommendation decisions.

2.1 Graph Neural Network

The Graph Neural Network (GNN) component is an adaptation of the GraphSAGE framework proposed by Hamilton et al. [3]. In the GraphSAGE framework, node embeddings are recursively computed up to a depth K. For a given node, a fixed number of neighbor nodes is uniformly sampled at each iteration, and their embeddings are aggregated (e.g., via mean pooling). The aggregated embedding is concatenated with the previously computed embedding of the node and passed through a non-linear layer to form the next embedding. The original GraphSAGE is used in a supervised setting, where each node has a label. During training, a batch of nodes is sampled, passed through the GNN network, and the final embedding is passed through a fully connected layer to predict the label.
To adapt the GraphSAGE framework, we made the following change: instead of predicting node labels, we predict edge labels. Specifically, given the final embeddings of a user and a recipe, we predict the rating using Cross Entropy Loss. This involves concatenating the two embeddings and passing them through a non-linear layer.
The initial embeddings are generated as follows: for each recipe, the recipe name and ingredients are concatenated and passed through the RecipeBERT model from HuggingFace (`https://huggingface.co/alexdseo/RecipeBERT`. RecipeBERT is a fine-tuned version of `bert-base-uncased` (proposed by Devlin et al. [2]) on the food-domain Recipe1M+ dataset (proposed by Marin et al. [5]). Initial user embeddings are generated by computing a weighted sum of the embeddings of recipes rated by the user, where the weights are determined by the ratings.

2.2 Multi-Armed Bandit & Contexual Bandit

Multi-Armed Bandit (MAB) problems represent a class of simplified Reinforcement Learning (RL) scenarios, characterized by the absence or simplicity of a state transition function. In these problems, each decision or 'arm pull' is independent of previous decisions since the effect of each decision does not impact future states. The central challenge in MAB problems is to model the reward distribution for each arm effectively. This focus on isolated decision-making emphasizes the exploration versus exploitation dilemma, where the agent must balance between choosing arms based on existing knowledge to maximize immediate rewards (exploitation) and trying arms selected less often to gather more information about their potential (exploration). This setting abstracts and distills the essence of decision-making under uncertainty, which is a key element in broader RL frameworks, but without the complexity of action dependent state transitions.

Contextual bandits extend the multi-armed bandit framework by introducing contexts which significantly enhance decision-making capabilities. Unlike traditional multi-armed bandits where each decision to pull an arm is independent of external information, contextual bandits incorporate a feature vector that provides additional information about the environment or situation. Each arm in a contextual bandit is associated with the same feature vector, allowing the algorithm to make more informed choices based on the current context. This model is particularly useful in applications where prior information is available and can affect the outcome of actions.

In this paper, we experiment with two commonly used Contextual Bandit algorithms, namely Thompson Sampling (TS) proposed by Agrawal et al. [1] and Linear Upper Confidence Bound (LinUCB) proposed by Li et al. [4].

Contextual Thompson Sampling (TS): This algorithm adopts a Bayesian approach to balance exploration and exploitation. It maintains a posterior distribution over the parameters θ of the reward model for each action. In each iteration, the algorithm samples θ from the posterior, computes the expected reward for each action, and selects the action with the highest reward. Formally, if $\mathbf{x}_a$ is the context vector for action a and θ_a is the sampled parameter for action a, the reward is estimated as:

$$r_a = \mathbf{x}_a^\top \theta_a,$$

and the action selected is:

$$a^* = \arg\max_a r_a.$$

After observing the reward for the chosen action, the posterior distribution of θ_a is updated based on the observed data.

Contextual Linear Upper Confidence Bound (LinUCB): LinUCB follows a deterministic approach to balance exploration and exploitation by computing an upper confidence bound for each action. The algorithm selects the action with the highest upper bound, which is a combination of the expected reward and a confidence interval. For each action a, the reward estimate is:

$$p_a = \mathbf{x}_a^\top \hat{\theta}_a + \alpha\sqrt{\mathbf{x}_a^\top \mathbf{A}_a^{-1}\mathbf{x}_a},$$

where $\hat{\theta}_a = \mathbf{A}_a^{-1}\mathbf{b}_a$ is the estimated parameter vector, $\mathbf{A}_a$ is the covariance matrix, $\mathbf{b}_a$ is the bias vector, and α controls the trade-off between exploration and exploitation. The action selected is:

$$a^* = \arg\max_a p_a.$$

The algorithm updates $\mathbf{A}_a$ and $\mathbf{b}_a$ for the selected action a based on the observed reward.

Both methods leverage contextual information to make decisions and adapt to changes in the environment, making them highly suitable for tasks like recipe recommendation in this work. In our paper, the **state vector** consists of the learned user and recipe embeddings, which encapsulate user preferences and recipe characteristics. The **reward** is defined as 1 if the predicted action aligns with the target action, and 0 otherwise. The **target action** is set to 1 (*recommend*) if the user's rating for the recipe is greater than or equal to 4, and 0 (*do not recommend*) otherwise. This setup allows the model to effectively learn and optimize for personalized recommendations.

2.3 Preferences and Similarity-Based Recommendations with LLM

The similarity-based recommendation method uses learned embeddings from the GraphSage model, with further selection throught a pre-trained large language model (LLM) to recommend recipes. This approach involves two stages:

1. **Similarity Scoring:** For a given user, their learned embedding is compared through cosine similarity as a dot product against all available recipe embeddings in the dataset. The top 10 recipes with the highest similarity scores are shortlisted for further processing acting as the most similar recipes.
2. **LLM suggested final recommendation:** The top 10 recipes selected are passed to a pretrained LLM and a final recommendation is generated by directing the LLM to find the most relevant recommendation after considering the preferences entered.

The LLM used in this work is **Google's Flan-T5**. Flan-T5 model is a fine-tuned version of the T5 (Text-to-Text Transfer Transformer) model [6], optimized for general NLP instruction and response tasks. The choice of Flan-T5 was motivated after trying similar prompts with several other lightweight LLMs such as Microsoft's Phi 3 and Phi 3.5 models and receiving poor results due to model diverging from prompts.
Using the LLM allows us to incorporate novel user preferences and provide novel suggestions while including users' past interaction histories.

3 Experiment

In this section, we describe the experiment, including implementation details (libraries used) and configurations (parameters and running environment).

3.1 Libraries

The following libraries were used for the implementation of our model:

- **NumPy (2.1.2):** For efficient numerical computations and matrix operations.
- **Pandas (2.2.3):** For data manipulation and preprocessing.
- **PyTorch (2.4.1):** For implementing and training the Graph Neural Network and Contextual Bandit models.
- **HuggingFace Transformers (4.46.3):** For using the RecipeBERT model to extract initial recipe embeddings.
- **Scikit-learn (1.5.2):** For evaluation metrics and additional utilities.
- **Scipy (1.14.1):** For advanced mathematical functions and optimizations.
- **Tqdm (4.66.5):** For tracking the progress of training and evaluation loops.
- **Requests (2.32.3):** For downloading external resources as needed.
- **Joblib (1.4.2):** For efficient serialization and parallel processing.
- **NetworkX (3.4.1):** For creating and managing graph data structures.
- **HuggingFace Hub (0.26.2):** For downloading and managing pre-trained models and tokenizers.
- **SymPy (1.13.3):** For symbolic mathematics used in formula validation and representation.
- **NumPy (2.1.2):** For efficient numerical computations and matrix operations.
- **Flask (3.0.0):** For creating the API and inline web interface.

These libraries provided the necessary tools and frameworks to implement, train, and evaluate the proposed model effectively.

3.2 Configurations

The initial feature vectors for recipes and users have a dimensionality of 768. The Graph Neural Network (GNN) component consists of two pairs of aggregation and non-linear encoding layers, with both encoding layers having an output dimension of 128. The number of neighbors sampled during training is set to 5. The model is trained for 20 epochs with a batch size of 256.
For the Contextual Bandit component, 80% of the user-recipe interactions are used for sequential training, while the remaining 20% is reserved for testing. The exploration parameter (α) for LinUCB is set to 1.0. Thompson Sampling dynamically balances exploration and exploitation by sampling from a Beta distribution for each action, so no explicit parameter was provided for this implementation.
For the LLM component, the recommendation system in addition to using cosine-similarity to shortlist recipes utilizes the Flan-T5-Large model with the T5Tokenizer from HuggingFace. Default generation parameters are used, including settings for token length, temperature, and sampling methods.
All experiments were implemented in PyTorch and executed on a single Mac M1 GPU with MPS support.

3.3 User Interface

We implemented a Flask-based API with an inline web page to allow interaction with the recommendation system. The UI enables users to:

- Enter a user ID, matched against unique user IDs from the dataset for embeddings.
- View past interactions of the selected user, showing the recipes they have rated and their corresponding ratings.
- Enter user preferences (optional for Thompson Sampling and LinUCB but mandatory for the LLM-based recommendation method).
- Choose one of three recommendation methods: **LinUCB**, **Thompson Sampling**, or **LLM Similarity**.

Upon submission, the API displays:

1. **Recommended recipes, tailored to the selected user and method.**
2. **Past recipes and ratings** for the user, when applicable.

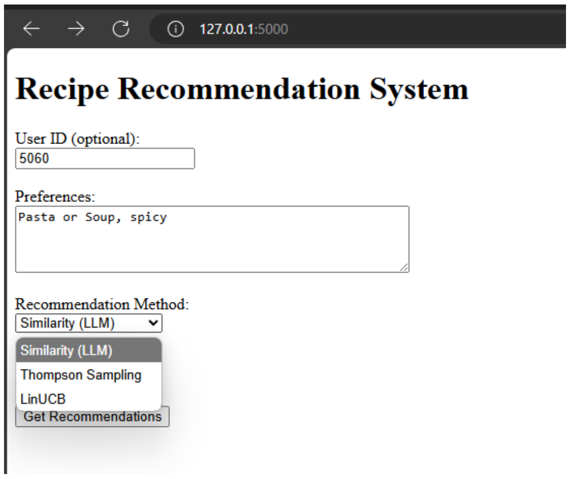
We show the UI in Fig 1, 2, 3, 4.

4 Dataset

In this paper, we utilized the dataset from Food.com, available at `https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions`. The dataset includes the following statistics:

- **Number of recipes:** 231,637
- **Number of users:** 226,570
- **Number of interactions:** 1,132,367

Each interaction consists of a user review and a corresponding rating.



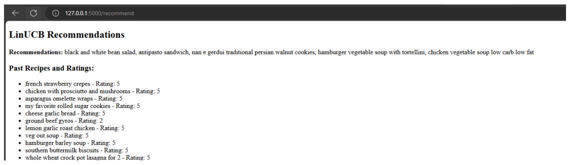
Recipe Recommendation System

User ID (optional): 5060

Preferences: Pasta or Soup, spicy

Recommendation Method: Similarity (LLM)

Figure 1: Home Page of App



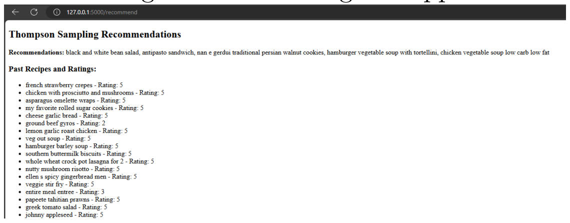
LinUCB Recommendations

Recommendations: black and white bean salad, antipasto sandwich, nan e gerdui traditional persian walnut cookies, hamburger vegetable soup with tortellini, chicken vegetable soup low carb low fat

Past Recipes and Ratings:

- french strawberry crepes - Rating: 5
- chicken with prosciutto and mushrooms - Rating: 5
- asparagus omelette wraps - Rating: 5
- my favorite rolled sugar cookies - Rating: 5
- cheese garlic bread - Rating: 5
- ground beef gyros - Rating: 2
- lemon garlic roast chicken - Rating: 5
- veg out soup - Rating: 5
- hamburger barley soup - Rating: 5
- southern buttermilk biscuits - Rating: 5
- whole wheat crock pot lasagna for 2 - Rating: 5

Figure 2: LinUCB result for sample user 5060



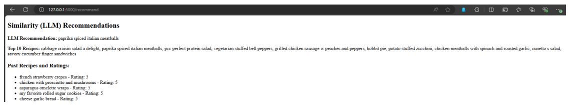
Thompson Sampling Recommendations

Recommendations: black and white bean salad, antipasto sandwich, nan e gerdui traditional persian walnut cookies, hamburger vegetable soup with tortellini, chicken vegetable soup low carb low fat

Past Recipes and Ratings:

- french strawberry crepes - Rating: 5
- chicken with prosciutto and mushrooms - Rating: 5
- asparagus omelette wraps - Rating: 5
- my favorite rolled sugar cookies - Rating: 5
- cheese garlic bread - Rating: 5
- ground beef gyros - Rating: 2
- lemon garlic roast chicken - Rating: 5
- veg out soup - Rating: 5
- hamburger barley soup - Rating: 5
- southern buttermilk biscuits - Rating: 5
- whole wheat crock pot lasagna for 2 - Rating: 5
- nutty mushroom risotto - Rating: 5
- ellen s spicy gingerbread men - Rating: 5
- veggie stir fry - Rating: 5
- entire meal entree - Rating: 3
- papeete tahitian prawns - Rating: 5
- greek tomato salad - Rating: 5
- johnny appleseed - Rating: 5

Figure 3: TS result for sample user 5060



Similarity (LLM) Recommendations

LLM Recommendation: paprika spiced italian meatballs

Top 10 Recipes: cabbage craisin salad a delight, paprika spiced italian meatballs, pcc perfect protein salad, vegetarian stuffed bell peppers, grilled chicken sausage w peaches and peppers, hobbit pie, potato stuffed zucchini, chicken meatballs with spinach and roasted garlic, cunetto s salad, savory cucumber finger sandwiches

Past Recipes and Ratings:

- french strawberry crepes - Rating: 5
- chicken with prosciutto and mushrooms - Rating: 5
- asparagus omelette wraps - Rating: 5
- my favorite rolled sugar cookies - Rating: 5
- cheese garlic bread - Rating: 5

Figure 4: LLM result for sample user 5060 with preference string - ”Spicy Italian”

5 Metrics

Below is the list of metrics used:

- **Recall@k:** Measures the proportion of relevant items that are recommended in the top k results.
- **Precision@k:** The proportion of recommended items in the top k that are relevant.
- **F1-Score:** Harmonic mean of Precision and Recall, balancing both metrics.

6 Baselines

Below is a list of the models that I tested on as well as their parameters:

- **ItemKNN (Item-based k-Nearest Neighbors)**
  - **Description:** ItemKNN is a traditional collaborative filtering algorithm that calculates the similarity between items based on user interaction histories. It recommends items to users by finding the k-nearest items (those with the highest similarity scores) that are frequently interacted with by similar users.
  - **Important Parameters:**
    - **k** = 100 (The number of nearest neighbors to consider for each item)
    - **shrink** = 0.0 (The shrinkage parameter for similarity computation, to regularize item similarities)

    - **epochs** = 300 (The number of epochs for training)
    - **train_batch_size** = 2048 (The batch size used for training)
    - **eval_batch_size** = 4096 (The batch size used for evaluation)

- **BPR (Bayesian Personalized Ranking)**
  - **Description:** BPR is a pairwise collaborative filtering method designed for optimizing ranking tasks in recommendation systems. It models the relative preferences between items and is optimized using a pairwise loss function.
  - **Important Parameters:**
    - **embedding_size** = 64 (Dimensionality of user and item embeddings)
    - **epochs** = 300 (Number of epochs for training)
    - **train_batch_size** = 2048 (Batch size for training)
    - **learning_rate** = 0.001 (Learning rate for optimization)
    - **eval_batch_size** = 4096 (Batch size used for evaluation)

- **NeuMF (Neural Matrix Factorization)**
  - **Description:** NeuMF combines both Generalized Matrix Factorization (GMF) and a Multi-Layer Perceptron (MLP) to capture both linear and non-linear patterns for collaborative filtering tasks. It models user-item interactions through the combination of GMF's matrix factorization and MLP's deep feature learning.
  - **Important Parameters:**
    - **mf_embedding_size** = 64 (The embedding size for the matrix factorization part of the model)
    - **mlp_embedding_size** = 64 (The embedding size for the MLP part of the model)
    - **mlp_hidden_size** = [128, 64] (The hidden layer sizes for the MLP component)
    - **dropout_prob** = 0.1 (The dropout rate for regularization)
    - **learning_rate** = 0.001 (The learning rate for optimization using the Adam optimizer)
    - **train_batch_size** = 2048 (The batch size used for training)
    - **epochs** = 300 (The number of epochs for training)
    - **mf_train** = True (Enable training for the matrix factorization part)
    - **mlp_train** = True (Enable training for the MLP part)
    - **use_pretrain** = False (Indicates that pre-trained embeddings are not used)

7 Results

Table 1 summarizes the results for the above baselines and our proposed methods (LinUCB and TS).

Method	**Precision**	**Recall**	**F1**
ItemKNN	0.00064	0.00527	0.00114
BPR	0.00259	0.01831	0.00454
NeuMF	0.06845	0.55323	0.12183
Ours (LinUCB)	0.88770	0.99970	0.94038
Ours (TS)	0.88760	1.00000	0.94045

Table 1: Comparison of Precision, Recall, and F1 Scores Across Methods

Please note, our method is not directly comparable to the baselines, as our precision, recall, and F1 metrics are computed across all test interactions using an 80-20 train-test split. In contrast, the baselines report precision@10 and recall@10 metrics. However, based on the received metrics, our method demonstrates highly promising performance.

8 Conclusion and Future Work

8.1 Conclusion

This project successfully combined Graph Neural Networks (GNNs) and Contextual Bandits to tackle the problem of recipe recommendation. Our approach demonstrated promising performance in terms of precision, recall, and F1 scores, highlighting the effectiveness of leveraging embeddings learned through GNNs alongside decision-making capabilities of Contextual Bandits.

8.2 Future Work

For future work, we propose the following directions for improvement and expansion:

- **Comparison with Contextual Bandit Baselines:** Compare our method against established contextual bandit baselines to provide a more relevant and fair performance evaluation.
- **Contrastive Learning:** Explore contrastive learning techniques to maximize the distance between positive and negative user-recipe pairs. Positivity and negativity will be determined based on user ratings, aiming to enhance the quality of learned embeddings.
- **Alternative LinUCB Actions:** Experiment with treating ratings (e.g., 1–5) as distinct actions in LinUCB. This can enable finer-grained decision-making and provide additional insights into user preferences.
- **Richer Explanations from LLMs:** Enhance the LLM-based recommendation system by generating richer explanations for each recommendation. We tried this initially but due to the lightweight LLMs used the token limit and prompt focus didn't allow us to achieve it reliably.
- **Scalability with Retrieval-Augmented Generation (RAG):** Integrate RAG techniques to handle large recipe datasets instead of using a similarity system to get top 10 recommendations, so that LLM can scale to datasets that exceed its token limit.
- **Fine-Tuning Flan-T5 for Recommendations:** Fine-tune Flan-T5 specifically for recipe recommendation tasks, optimizing it for preference match.
- **Hybrid Systems:** Combine LLM-based recommendations with contextual bandit policies to create a hybrid system. This would allow user-specific recommendations with exploration of new recipe options providing a strong method of solving multiple challenges.

These efforts will deepen our understanding of the proposed model and its potential for further refinement, paving the way for a more robust and versatile recommendation system.

References

[1] Shipra Agrawal and Navin Goyal. Thompson sampling for contextual bandits with linear payoffs. *30th International Conference on Machine Learning, ICML 2013*, 09 2012.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2019.

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. 2018.

[4] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, WWW '10, page 661–670. ACM, April 2010.

[5] Jorge Marin, Vladislav Karpukhin, Sebastien Noury, Neil Alldrin, Sachin Menon, Maria Vasileva, and Kavita Bala. Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5975–5984, 2019.

[6] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.