COMP9727 Recommender Systems

Feature Selection Process

The training data mainly consist of two parts: recipes information and user rating.

After observing the training data and testing data, we can conclude the problem as predicting the *specific rating* of *specific recipe* by *specific user* at a *specific time*.

As a result, we should realize that this problem is restricted by multiple variables, consisting user, recipe and time.

It is not a good idea to directly use the user id and recipe id as the model input, as ID is only an index and cannot provide valid information for model. Thus, it is essential to obtain user embedding and the recipe embedding from the given data.

For user embedding and time information, I will discuss them in the following sections as they are tightly associated with the model design.

Thus, the key problem here is to select the features from recipes information. As neural networks or MLP are mostly black boxes, I included most features for the neural network approach except submitter id and submitting date.

Data Processing Procedure

For data preprocessing strategies, we mainly have the following challenges:

- how to process the list data
- how to process the data in natural language

For the first challenge, I used multi-hot encoding for the `tags` data in `recipes_info`, as there are only a limited amount of tags (511) in the whole file, which is a tolerable size for input feature.

However, this approach cannot solve the encoding problem for fields such as `ingredients` and `steps` as the unique counts for those fields are outrageous, and different recipes share few similarity across the dataset, which made multi-hot encoding not an ideal solution for this case.

To solve this problem, along with the encoding for `name` and `description` fields, I think it's not a bad idea to use natural language processing model to obtain sentence feature vector.

Here, I used SentenceBERT as the sentence embedding network.

> 这里开始没做过

I fine tuned the pretrained SentenceBERT model using the corpus made from the extracted sentences in the recipe dataset.

<<<<< 一直到这里

The model helped me extract features from `name`, `description`, `ingredients`, and `steps` fields after text cleaning.

For other fields such as `nutrition` and `minutes`, I used them directly as the input, along with the batch normalization strategy to prevent overfitting.

Besides, I also discovered that the recipe IDs are discrete, which is not good for indexing data for the data pipeline. Though data structures such as HashMap can enable $O(1)$ retrieval, it will still result in some constant level performance cost. As a result, I performed discretization for the recipe ids so that they can be indexed as array.

Model Design

Model Input

Before talking about model design, I think it is important to clarify the model input. According to our problem, it is obvious that we have to include the recipe embedding C and user interest embedding U. The problem is how to handle the timestamp efficiently.

User-date Embedding

Here, I decided to integrate the timestamp with user interest embedding. Notate $U(t)$ as the user interest embedding at time t. It is natural to obtain the user interest embedding according to time by using the past ratings for recipes by that specific user. Define $U(t)$ as the weighted average of recipe embedding for all recipes rated by the user before time t,

$$U(t) = \frac{1}{n} \sum_{R_i.t<t} C_i \odot W(R_i)$$

in which $\boldsymbol{R}$ represents the whole set of rating by that user, $R_i.t$ and C_i represents the rating timestamp and corresponding recipe embedding of rating R_i, $W(\cdot)$ represents the rating weight function, and R_i represents the score of that rating.

Dynamic Mini-batch Approach

The above design seems nice, but it also introduced new problems. By brute-force, the complexity of computing all user-date embedding across the dataset will reach $O(\sum_i |U_i|^2)$, in which $|U_i|$ represents how many ratings each user U_i produce. Computing a single user-date embedding is already computationally intensive, and the design also does not support mini-batch training, which may also cause long training time and slow converging process during stochastic gradient descent.

To solve this problem, I introduced a dynamic mini-batch approach. Instead of calculating the user-date embedding one at a time, we can calculate the user-date embedding for each user at the same time, which will greatly optimize the time complexity from $O(\sum_i |U_i|^2)$ to $O(|U|)$.

As each user only has limited rating samples in the dataset, we can concat the rating data of a specific user as following

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ \cdots \\ C_{n-1} \end{bmatrix} \odot W \left(\begin{bmatrix} R_1 \\ R_2 \\ R_3 \\ \cdots \\ R_{n-1} \end{bmatrix} \right)$$

in which C_i and R_i represents the recipe embedding and rating score correspondingly.

By constructing an upper triangular matrix and using boardcasting mechanism in numpy and PyTorch, we can get a beautiful result:

$$\begin{bmatrix} C_1 \odot W(R_1) \\ C_2 \odot W(R_2) \\ C_3 \odot W(R_3) \\ \cdots \\ C_{n-1} \odot W(R_{n-1}) \end{bmatrix} \odot \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n-1} \\ & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n-1} \\ \vdots & & \ddots & \ddots & \vdots \\ & & & \ddots & \frac{1}{n-1} \\ & (0) & & & \frac{1}{n-1} \end{bmatrix}$$

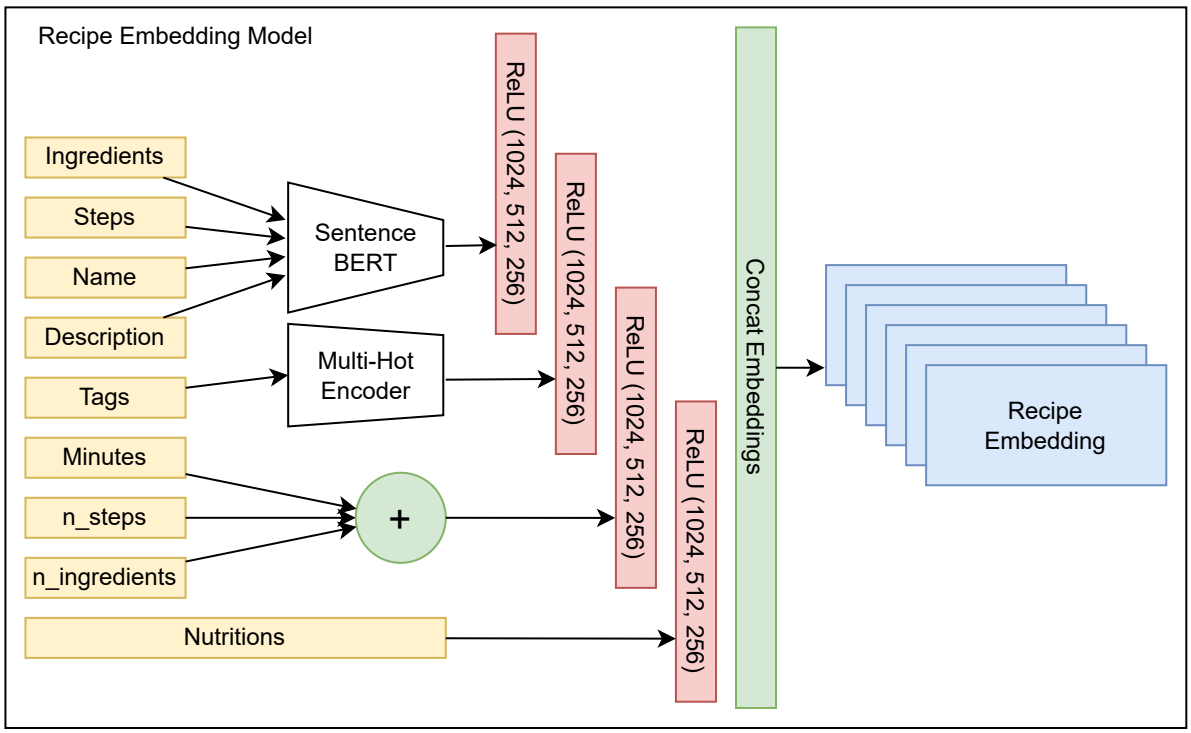
$$= \begin{bmatrix} C_1 \odot W(R_1) & \frac{1}{2}C_1 \odot W(R_1) & \frac{1}{3}C_1 \odot W(R_1) & \cdots & \frac{1}{n-1}C_1 \odot W(R_1) \\ & \frac{1}{2}C_2 \odot W(R_2) & \frac{1}{3}C_2 \odot W(R_2) & \cdots & \frac{1}{n-1}C_2 \odot W(R_2) \\ \vdots & & & \ddots & \ddots & \vdots \\ & & & & \ddots & \frac{1}{n-1}C_{n-2} \odot W(R_{n-2}) \\ & (0) & & & & \frac{1}{n-1}C_{n-1} \odot W(R_{n-1}) \end{bmatrix}$$

After obtaining the sum by column, we can obtain the user-date embedding for all date of a specific user,

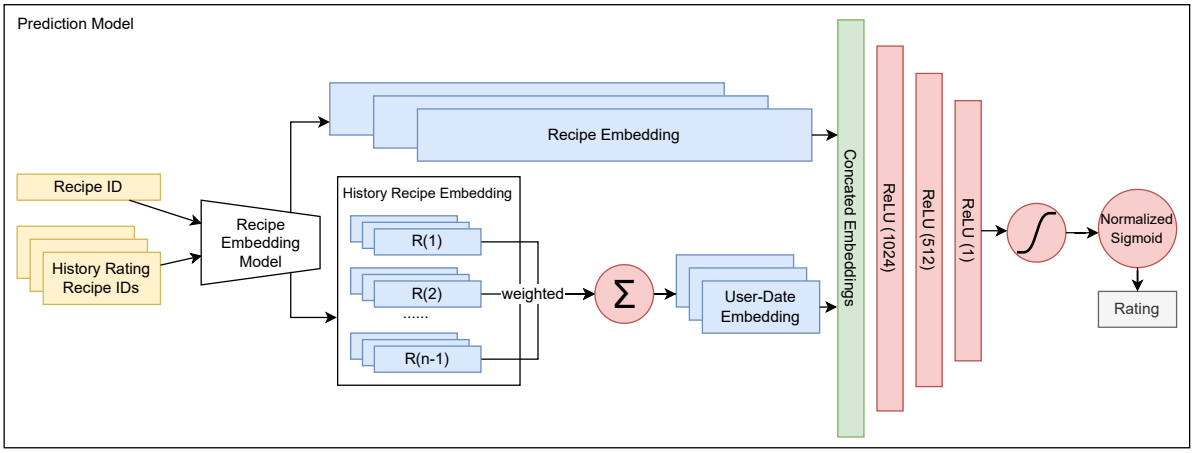
$$\begin{bmatrix} \vec{0} \\ \sum_{i=1}^{1} C_i \odot W(R_i) \\ \frac{1}{2}\sum_{i=1}^{2} C_i \odot W(R_i) \\ \frac{1}{3}\sum_{i=1}^{3} C_i \odot W(R_i) \\ \cdots \\ \frac{1}{n-1}\sum_{i=1}^{n-1} C_i \odot W(R_i) \end{bmatrix} = \begin{bmatrix} U(R_1.t) \\ U(R_2.t) \\ U(R_3.t) \\ U(R_4.t) \\ \cdots \\ U(R_n.t) \end{bmatrix}$$

Model Structure

The following diagram shows the recipe embedding model structure.



Using the above module, here comes the complete model structure.



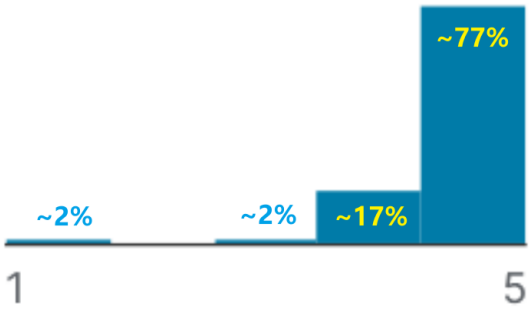
Overall, this method use the MLP approach to predict the user rating, and use MSE (Mean Square Error) as the loss function to optimize the model.

Normalized Sigmoid

In the last section, I introduced the model structure, in which we can see that the last two stages of the model consist of sigmoid and normalized sigmoid.

The first sigmoid function is used to map the feature obtained from the last model stage from $\mathbb{R}$ to $(0, 1)$.

However, mere sigmoid mapping strategy is not enough in this case. After experiments, I discovered that the original rating score distribution highly affects the training process.



To solve this problem, I introduced a non-linear score mapping strategy to the model, the normalized sigmoid mapping module.

$$y = \frac{x - kx}{k - 2k|x| + 1}, k \in [-1, 1]$$

Here, I picked $k = -0.7$ as a hyperparameter.

MISC Tricks

- To prevent overfitting, batch normalization and weight decay are used.
- Cosine Annealing for learning rate scheduler

Reference

- SentenceBERT: https://arxiv.org/abs/1908.10084
- Normalized Sigmoid: https://dhemery.github.io/DHE-Modules/technical/sigmoid/
- Cosine Annealing: https://arxiv.org/abs/1608.03983v5