INTERPRETABLE RELATIONAL REPRESENTATIONS FOR FOOD INGREDIENT RECOMMENDATION SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Supporting chefs with ingredient recommender systems to create new recipes is challenging, as good ingredient combinations depend on many factors like taste, smell, cuisine style, texture among others. There have been few attempts to address these issues using machine learning. Importantly, useful models do obviously need to be accurate but importantly – especially for food professionals – interpretable. In order to address these issues, we propose the Interpretable Relational Representation Model (IRRM). The main component of the model is a key-value memory network to represent relationships of ingredients. We propose and test two variants of the model. One can learn latent relational representations over a trainable memory network (Implicit model), and the other can learn explainable relational representations over a pre-trained memory network that integrates an external knowledge base (Explicit model). The relational representations resulting from the model are interpretable – they allow to inspect why certain ingredient pairings have been suggested. The Explicit model additionally allows to integrate any number of manually specified constraints. We conduct experiments on two recipe datasets, including CulinaryDB with 45,772 recipes and Flavornet with 55,001 recipes, respectively. The experimental results show that our models are both predictive and informative.

1 INTRODUCTION

Due to the increasing availability of food related content – data mining and machine learning methods play an increasingly prominent role in food preference modelling, food paring which discover good ingredient combinations, and new recipe generation. Solving these tasks is non-trivial, since the goodness of ingredient combinations depends on many factors like taste, smell, cuisine, texture, and their culture.

Ahn et al. (2011) detected that the number of shares of flavor molecules between ingredients is one of important factors for food paring. They found Western cuisines show tendency to use ingredient pairs that share many flavor compound, while East Asian cuisines tend to avoid compound sharing ingredients. Using this idea, Garg et al. (2017) developed a rule-based food paring system which ranks ingredients based on the number of shares of flavor molecules. Recently, Park et al. (2019) suggested a neural network approach based on flavor molecules and co-occurrence of ingredients in recipes. These approaches focus on one-to-one food paring. There are also some researches related with many-to-one paring. De Clercq et al. (2016) proposed the recipe completion task which assumes the situation for preparing a meal using some leftover ingredients in a refrigerator, and then adopted a recommender style approaches with Matrix Factorization. Although efforts have been made to detect good ingredient combinations, there is still no method that can interpret the reasons why suggested pairs are good. Our work is targeted at interpretable recommendation systems for the food pairing, that is, given some preselected ingredients by an user, the recommender suggests top-N ingredients from candidates. Therefore, our models support both one-to-one and many-to-one paring. For example, suppose an user selects *apple* and *chocolate* as the preselected ingredients, our recommender suggests some good paired ingredients (e.g. *cinnamon*) with reasons (e.g. *cinnamon* is good for *apple* and *chocolate* in terms of their flavor affinity).

In this paper, we propose the Interpretable Relational Representations Model (IRRM) in two variants to address food pairing. The model features a key-value memory network (Sukhbaatar et al.

(2015), Miller et al. (2016)) to represent relationships of ingredients. One variant of the model is trained to learn latent relational representations over the trainable memory network (Implicit model), and the other can learn explainable relational representations over the pre-trained memory network which integrates an external knowledge base (Explicitly model). The relational representations are interpretable and can be queried as to the reasons why the ingredients have been suggested. The Explicit model can integrate any number of constraints which can be decided manually based on the characteristics of the desired recommender system. Our contributions are as follows:

1. We model ingredient pairing as a general recommendation task with implicit feedback.
2. We introduce the Interpretable Relational Representations Model and it's two variants: Implicit and Explicit. Both of which can learn pair specific relational representations (vectors) for one-to-one (i.e. ingredient to ingredient) and many-to-one (ingredient-set to ingredient) food pairing tasks. The relational vectors are also interpretable.
3. We propose a training procedure to learn one-to-one and many-to-one relationships effectively using recipes.
4. We evaluate our proposed models in the Recipe Completion Task and the Artificial Food Paring Task on the CulinaryDB and the Flavornet datasets. Our proposed approaches demonstrate competitive results on all datasets, outperforming many other baselines.
5. We perform qualitative analysis. The results presents our proposed Explicit model is capable of unraveling hidden ingredients structures within recipes

2 Problem Definition

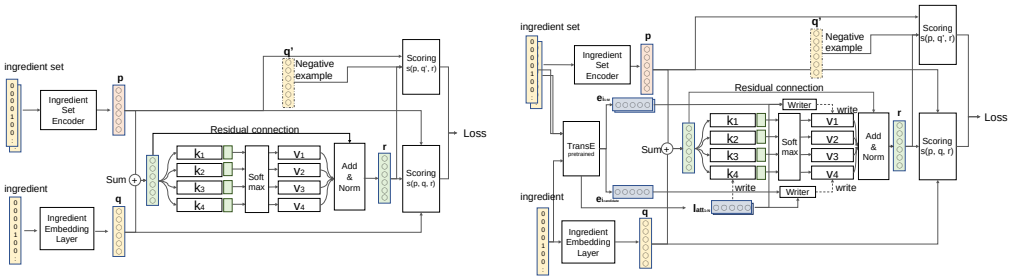
We first introduce the notations used throughout this paper. We model ingredient pairing as a recommendation scenario with implicit feedback (Huang et al. (2018), Tay et al. (2018)), where the only concern is whether a user has interacted with an item and the system infers the item that user will interact next based on the interaction records of the user.

In our food ingredient recommendation system, we use recipes instead of the interaction records. Let $\mathcal{I}$ denote a set of ingredients and $\{i_1, \ldots, i_M\}$ denote a preselected ingredient set, where $i \in \mathcal{I}$ is the ingredient and M is the number of ingredients. We call $\{i_1, \ldots, i_M\}$ preselected ingredients or a preselected ingredient set in this paper. Next, let $\mathcal{I}_{candidate}$ denote a set of candidate ingredients which depends on each preselected ingredient set, that is, $\mathcal{I}_{candidate} = \mathcal{I} - \{i_1, \ldots, i_M\}$. We call $\mathcal{I}_{candidate}$ candidate ingredients or a candidate ingredient set. In addition, we assume that a knowledge base (KB) of ingredients is also available and the KB contains factors which are related to why some ingredients are good for combinations. A KB is defined as a set of triplets over a entity set $\mathcal{E}$ and a relationship set $\mathcal{L}$. A KB triplet $\langle e_i, l, e_a \rangle$ composed of two entities $e_i, e_a \in \mathcal{E}$ and a relationship $l \in \mathcal{L}$, where e_i is an ingredient (e.i. $e_i \in \mathcal{I}$) and l is an attribute and e_a is the attribute value. For instance, $\langle$*apple, flavorMolecule, (-)-Epicatechin*$\rangle$ denotes that *apple* contains the *(-)-Epicatechin* flavor molecule.

Based on these preliminaries, we define the food ingredient recommendation task. Given a preselected ingredient set $\{i_1, \ldots, i_M\}$ and candidate ingredients $\mathcal{I}_{candidate}$, we would like to infer the top-N ingredients from $\mathcal{I}_{candidate}$.

3 Recommendations with Memory Network

In this section, we introduce our IRRM architectures. We start with the Implicit model that consists of a trainable key-value memory network. We then augument the Implicit model using a key-value memory network which integrates pre-trained entity and relationship vectors with ingredient attributes in the KBs – we call this extention the Explicit model. The overall model architecture is described in Figure 1. Our models aim to model preselected ingredient sets, ingredients, and relationships of pairs between a preselected ingredient set and an ingredient. The input of our models is a pair between a preselected ingredient set and an candidate ingredient $i_{candidate} \in \mathcal{I}_{candidate}$, and the output is a score of the pair. In the prediction phase, our recommender uses these models to rank $\mathcal{I}_{candidate}$.



(a) Implicit model

(b) Explicit model

Figure 1: IRRM architectures

3.1 Ingredient Embedding Layer and Ingredient Set Encoder

Ingredients are represented as vectors encoded via one-hot encoding (corresponding to a unique index key belonging to each ingredient). At the embedding layer,this one-hot encoded vector is converted into a low-dimensional real-valued dense vector representation which is multiplied with the embedding matrices $\boldsymbol{Q} \in \mathbb{R}^{d \times |\mathcal{I}|}$ – which stores ingredient embeddings. d is the dimensionality of the ingredient embeddings while $|\mathcal{I}|$ is the total number of ingredients. $i_{candidate}$ is converted to $\boldsymbol{q}$ using this embedding layer. On the other hand, the preselected ingredients $\{i_1, \ldots, i_M\}$ is encoded by the Ingredient Set Encoder (Figure 6). At first, each ingredient i_j is converted to a vector using the Ingredient Embedding Layer (same as $i_{candidate}$). As a result, $\boldsymbol{i}_j \in \mathbb{R}^d$ vectors are generated. The sum of these vectors is converted to the ingredient set vector $\boldsymbol{p}$ using a feed-forward network with a single hidden layer, followed by Layer Normalization.

3.2 Relation Encoder

Tay et al. (2018) introduced the LRAM (Latent Relational Attentive Memory) module in order to generate the latent relational vectors between user-item interactions, which gives insight and interpetability of the model. In order to improve performance, we expand this module by adding a residual connection, followed by Layer Normalization. This idea is inspired by Vaswani et al. (2017).

Given the pair of a preselected ingredient set vector and a candidate ingredient vector, $\langle \boldsymbol{p}, \boldsymbol{q} \rangle$, the Relation Encoder first applies $\boldsymbol{s} = \boldsymbol{p} + \boldsymbol{q}$ to generate the joint embedding of $\boldsymbol{p}$ and $\boldsymbol{q}$. The generated vector $\boldsymbol{s} \in \mathbb{R}^d$ is of the same dimension of $\boldsymbol{p}$ and $\boldsymbol{q}$. Note that while other functions like the element-wise multiplication and the multi-layered perceptron $MLP(\boldsymbol{p}, \boldsymbol{q})$ are also available, we found that the addition performs better. This joint embedding $\boldsymbol{s}$ is used as the input of the memory network. The attention vector $\boldsymbol{a} \in \mathbb{R}^d$ is a vector of importance weights over keys which are represented as the key matrix $\boldsymbol{K} = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_N]^T \in \mathbb{R}^{N \times d}$, where N is the number of key-value pairs in the memory network and $\boldsymbol{k}_j \in \mathbb{R}^d$ is a key vector. Each element of the attention vector $\boldsymbol{a}$ can be defined as $a_j = \boldsymbol{s}^T \boldsymbol{k}_j$, where $a_j \in \mathbb{R}$. In order to normalize the attention vector $\boldsymbol{a}$ to a probability distribution, we use the Softmax function:$Softmax(a_j) = \frac{\exp(a_j)}{\sum_{n=1}^{N} \exp(a_n)}$. We generate the vector $\boldsymbol{m} = \sum_{n=1}^{N} Softmax(a_n)\boldsymbol{v}_n$ as the summation of weighted value vectors which are represented as the value matrix $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N]^T \in \mathbb{R}^{N \times d}$. Finally, in order to generate the relational vector $\boldsymbol{r}$, $\boldsymbol{m}$ is added with the joint embedding $\boldsymbol{s}$ and Layer Normalization is applied like $\boldsymbol{r} = LayerNorm(\boldsymbol{s} + \boldsymbol{m})$.

3.2.1 The Explicit Model

In order to improve interpretability and predictive performance, we incorporate ingredient attribute information from a given KB into the memory network. Inspired by recent works which integrate a memory network with external memories (Huang et al. (2018)), we propose the Explicit Relational Encoder. Instead of the trainable key matrix $\boldsymbol{K}$ and value matrix $\boldsymbol{V}$, we use pre-trained vectors which are trained over the KB in advance. The key and value matrix are frozen in our explicit model

training. Given a pair of a preselected ingredient set $\{i_1, \ldots, i_M\}$ and an candidate ingredient $i_{candidate}$, $\{i_1, \ldots, i_M, i_{candidate}\}$ is converted into the entity vectors using the KB embeddings which provide the entity vectors $\boldsymbol{e} \in \mathbb{R}^{d^{KB}}$ and the relationship vectors $\boldsymbol{l} \in \mathbb{R}^{d^{KB}}$. Note that in case of $d^{KB} \neq d$, we convert the joint embedding $\boldsymbol{s} \in \mathbb{R}^d$ into $\boldsymbol{s}' \in \mathbb{R}^{d^{KB}}$ and the relational vector $\boldsymbol{r} \in \mathbb{R}^{d_{KB}}$ into $\boldsymbol{r}' \in \mathbb{R}^d$ with linear projections. We use the TransE (Bordes et al. (2013)) for the KB embeddings. Because, given the triplet $\langle e_i, l_{att}, e^i_{att} \rangle$, the TransE can learn entity vectors and relationship vectors to follow $\boldsymbol{e}^i_{att} = \boldsymbol{e}_i + \boldsymbol{l}_{att}$. KB relationships usually correspond to attribute types of entities, so we use the notation l_{att} as the attribute type and e^i_{att} as its value. Hence, we set the key matrix like the following:

$$\boldsymbol{K} = [\boldsymbol{l}_{att_1}, \ldots, \boldsymbol{l}_{att_N}]^T \tag{1}$$

where N depends on the number of attribute types which you want to integrate into and $\boldsymbol{K}$ is the constant matrix through training. And the value matrix is initialized for the each pair based on:

$$\boldsymbol{v}_{att_j} = \sum_{i \in \{i_1, \ldots, i_M, i_{candidate}\}} \boldsymbol{e}^i_{att_j} = \sum_{i \in \{i_1, \ldots, i_M, i_{candidate}\}} (\boldsymbol{e}_i + \boldsymbol{l}_{att_j}) \tag{2}$$

$$\boldsymbol{V} = [\boldsymbol{v}_{att_1}, \ldots, \boldsymbol{v}_{att_N}]^T \tag{3}$$

Since there are many one-to-multiple relations in KB, e.g. the apple has multiple flavor molecules in case the attribute type is $flavorMolecule$, the entity vector e_{att} on the KB embeddings is not the ingredient specific vector. Hence, we use $\boldsymbol{e}_i + \boldsymbol{l}_{att}$ instead of using e_{att}.

3.3 SCORE FUNCTION

Finally, we define our score function as the relationship between the preselected ingredient set vector $\boldsymbol{p}$, the candidate ingredient vector $\boldsymbol{q}$, and the relational vector $\boldsymbol{r}$:

$$s(\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{r}) = CosSim(\boldsymbol{p}, \boldsymbol{q}) + CosSim(\boldsymbol{p} + \boldsymbol{q}, \boldsymbol{r}) \tag{4}$$

where $CosSim$ is the cosine similarity, and good pairs will have high scores and vice versa.

We show the geometric differences between our score function and others in Figure 2. While Figure2(a) tries to place the ingredient set and each ingredient into the same spot in vector space, Figure2(b) learns to fit the ingredient set and each ingredient with the relational vector $\boldsymbol{r}$, which concepts come from translation between user and item in the domain of collaborative filtering. These score functions are suggested in previous works. Tay et al. (2018) reported $-||\boldsymbol{p} + \boldsymbol{r} - \boldsymbol{q}||$ could achieve better performance than $-||\boldsymbol{p} - \boldsymbol{q}||$ in the domain of collaborative filtering, which is the user-item based recommender. However, in the food pairing task, the results was the opposite. It seems if we use $-||\boldsymbol{p} + \boldsymbol{r} - \boldsymbol{q}||$, because of $\boldsymbol{r}$, $\boldsymbol{p}$ and $\boldsymbol{q}$ can not be trained properly. As $\boldsymbol{r}$ approaches $\vec{0}$, the performance is improved, so the representation of $\boldsymbol{r}$ cannot be learned. Since the ingredient set is a mixture rather than a list of ingredients, and it is also an ingredient like a kind of processed ingredient, this result is reasonable. In addition, since our network structure is symmetric with respect to $\boldsymbol{p}$ and $\boldsymbol{q}$, we adopt the score function which is a symmetric bilinear form (Figure2(c)). Our score function tries to place the ingredient set $\boldsymbol{p}$ and each ingredient $\boldsymbol{q}$ into the same spot in vector space and rotate the relationship between $\boldsymbol{p}$ and $\boldsymbol{q}$ closer to the same place as a relational vector $\boldsymbol{r}$ which is represented using attributes from KB.

3.4 OBJECTIVE FUNCTION

Our objective function is defined as:

$$L = \sum_{x=1}^{Batch} \sum_{y=1}^{Pos} -log[\frac{\exp(\frac{s(\boldsymbol{p}_x, \boldsymbol{q}_y, \boldsymbol{r}_{xy}) - \lambda}{\tau})}{\exp(\frac{s(\boldsymbol{p}_x, \boldsymbol{q}_y, \boldsymbol{r}_{xy}) - \lambda}{\tau}) + \sum_{z=1}^{Neg} \exp(\frac{s(\boldsymbol{p}_x, \boldsymbol{q}^z_y, \boldsymbol{r}_{xy})}{\tau})}] \tag{5}$$

where λ is the margin that separates the golden pairs and corrupted pairs, τ is the temperature parameter, $Batch$ is the mini-batch size, Pos is the number of positive examples, Neg is the number

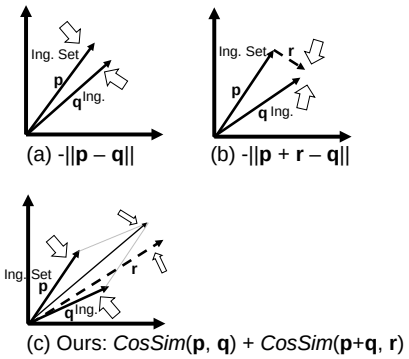


Figure 2: Geometric comparisons of score functions.

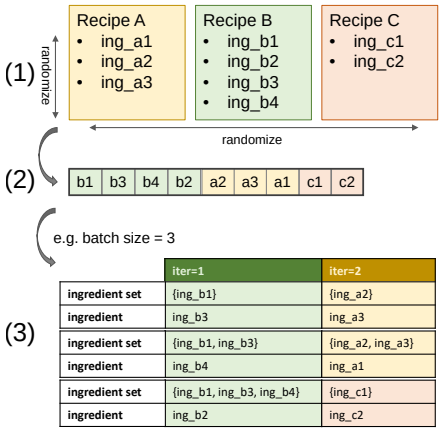


Figure 3: How to generate pairs for mini-batches.

of negative examples for each positive example, and the score function for negative examples take the same relational vector as their positive example. In oder to define this objective function, we refer to the Batch All loss (Hermans et al. (2017)) for the effective triplet loss in Metric Learning and the additive margin in the softmax loss (Huang et al. (2018)). Note that while the hinge loss is also available, we found the softmax loss is better performance and stable for customizing the definition of score functions.

3.5 Training

In this section, we propose how to generate mini-batches from recipes in our training scheme, which is inspired by the Session-Parallel mini-batches (Hidasi et al. (2016)) and the Link Prediction based on Graph Neural Networks (Zhang & Chen (2018)). In the Graph domain, nodes are embedded by neighbor nodes which are used as its context, e.g. local enclosing subgraphs (Zhang & Chen (2018)). Since each recipe is like the local enclosing subgraph, we generate a mini-batch to have the recipe related pairs parallelly. Beforehand we eliminate ingredients which has no attributes and then remove recipes which size is less than 1. Using pre-processed recipes, we train our models in the following steps (Figure 3): At first, we randomize the order of recipes and its ingredients (Figure 3(1)), and then generate the sequence of ingredients from recipes (Figure 3(2)). After that, we generate pairs between a preselected ingredient set and a candidate ingredient from the sequence like Figure 3(3) and loosely make pairs from same recipes belong to the same mini-batch. Next, we sample additional positive examples as necessary which are taken from other recipes randomly. Finally, we sample negative examples randomly. In the end, mini-batches are generated. Note that although we also trained our models without the recipe restriction, the performance was degraded in both one-to-one and many-to-one pairs.

4 Evaluation

We evaluated the IRRMs, the Implicit model and the Explicit model, on two tasks against standard baselines and we analyzed the relational representations in terms of interpretability. We propose two tasks for evaluation

For evaluation, we first use the same ranking procedure as in De Clercq et al. (2016), namely the *Recipe Completion Task*. In the Recipe Completion Task, the recommender ranks all ingredients to predict one randomly eliminated ingredient, and the remaining ingredients of each recipe are used as input to the model – which is the preselected ingredient set in our definition. We adopt three evaluation metrics for this task (same as the previous work); Mean Rank: the mean position of the eliminated ingredient in the ranked ingredients, Hit Ratio(HR)@10: the proportion of correct ingredients ranked in the top 10, and ROC-AUC(AUC): the mean area under the ROC curve. This

task is useful as it allows us to measure the basic performance of our model in the same setting as existing baselines (for many-to-one pairing) and against ground truth data.

In order to understand the model performance further, we also prepare a another prediction task, which we call *Artificial Food Pairing Task*. In this task we generate pairs from existing recipes based on ingredient co-occurrences. Given some ingredients as a preselected ingredient set and a candidate ingredient, pairs that occur in any of recipes are used as a positive example, otherwise it is a negative candidate example. The task of the Artificial Food Pairing consists of positive pairs and negative pairs where a same preselected ingredient set is always used for both positive and negative pairs but positive and negative examples are randomly taken from its candidates with a pre-specified the number and ratio. In this task, a recommender predicts whether pairs are positive or negative examples based on the ranking. We use this task for more detailed analysis. In this task, we use MAP@10: the mean average precision in top-10 and AUC as metrics.

We evaluate the proposed models against some baselines on two datasets. The first dataset is taken from the CulinaryDB (Bagler (2017)), and the second dataset is from Flavornet (Ahn et al. (2011)). Both datasets contain the recipe set which consist of the list of ingredients and the cuisine categories (e.g. Chinese, Japanese) and the ingredient sets which consist of the names, the food categories (e.g. Meat, Fruit), the flavor molecules the ingredient has. The statistics of datasets shows in Table 4. Before training models, the recipe set are divided into a train, a validation and a test set. On the other hand, we generate triplets (Figure 7) from whole ingredient set in order to train TransE, 172,207 triplets for CulinaryDB and 40,952 triplets for Flavornet.

4.1 Implementation

We implemented our models in PyTorch[1] on a Linux machine. For tuning the hyperparameters, we select the model that performs best on the validation set based on the precision metric and report the results on the test set. All models are optimized by the Adam optimizer (Kingma & Ba (2015)), and are trained until convergence, the maximum is 35 epochs. The dimensionality d and $d^{KB}$is tuned amongst $\{64, 128, 200\}$, the batch size $Batch$ is tuned $\{16, 32, 64\}$, the number of positive examples Pos is tuned $\{1, 2, 3, 5, 10\}$, the number of negative examples is tuned $\{1, 2, 3, 5, 10\}$, the learning rate is tuned $\{0.1, 0.01, 0.001\}$, the margin λ is tuned $\{0.1, 0.2, 0.25, 0.5\}$, the temperature τ is tuned $\{0.5, 1, 10\}$. Since CulinaryDB has 4 attribute types and Flavornet has 3 attribute types, we use the memory size $N_{CulinaryDB} = 4$ and $N_{Flavornet} = 3$ respectively in our Implicit/Explicit models.

For most datasets, we found the following hyperparameters work well: $d = d^{KB} = 128$, $Batch = 64$, $Pos = 1$ is for Recipe Completion Task, $Pos = 2$ is for Artificial Food Pairing Task, $Neg = 5$, learning rate $= 0.001$, $\lambda = 0.2$, and $\tau = 1$. Ingredients with less than 1 occurrence in the train set are learned as the UNK ingredient.

4.2 Baselines

We compare our Implicit/Explicit models to the following baselines.

- **FREQ**: Frequency predictor that always recommend the most frequently used ingredients of the training recipes. Despite its simplicity it is often a strong baseline in certain domains.
- **PMI**: Recommender based on the PMI score.
- **TFIDF**:Recommender based on the TF-IDF score.
- **NMF**: Non-negative matrix factorization based recommender which is suggested by De Clercq et al. (2016). The model is trained using the train and the validation recipes. It is implemented by ourselves.

4.3 Results and Discussion

Predictive performance Table 1 shows the results on all datasets for all compared baselines in Recipe Completion Task. While both our Implicit and Explicit models outperform all counterparts

[1]https://pytorch.org/

Table 1: Experimental results on the Recipe Completion Task.

Datasets	CulinaryDB			Flavornet		
Metrics	Mean Rank	HR@10	AUC	Mean Rank	HR@10	AUC
FREQ	473.5	0.119	0.605	257.9	0.131	0.621
PMI	612.5	0.055	0.527	346.9	0.066	0.531
TFIDF	478.5	0.055	0.598	261.3	0.055	0.613
NMF	57.0	**0.435**	0.900	36.3	0.479	0.896
IRRM(Implicit)	**35.9**	0.391	**0.943**	**28.1**	**0.485**	**0.926**
IRRM(Explicit)	**35.4**	0.397	**0.944**	**28.3**	0.480	**0.925**

Table 2: Experimental results on the Artificial Food Pairing Task (CulinaryDB).

Pair type	one-to-one		two-to-one		three-to-one	
Metrics	MAP@10	AUC	MAP@10	AUC	MAP@10	AUC
NMF	0.638	0.702	0.604	0.786	0.601	0.794
W/O-RELENC	0.574	0.736	0.494	0.731	0.499	0.746
IRRM(Implicit)	0.839	**0.904**	0.755	**0.921**	**0.792**	**0.925**
IRRM(Explicit)	**0.842**	**0.906**	**0.759**	**0.922**	**0.790**	**0.926**

on all metrics usually with a clear margin, our Explicit model is approximately same performance as the Implicit model. On the other hand, NMF is the best in baselines. Performance is good since the Recipe Completion task requires finding only one missing ingredient - eventhough there should be multiple correct ingredients in the real situation. It appears that our approaches are effective for many-to-one pairs which are constrained by recipes.

In order to evaluate the pairing skills from one-to-one to many-to-one, we use the Artificial Food Paring Task. We report the results of one-to-one, two-to-one, and three-to-one. We sample positive examples and negative examples according to three ratio settings like $(Pos, Neg) \in \{(20, 80), (50, 50), (80, 20)\}$. Beforehand we classify all ingredients into two categories: $\{Low, High\}$ depending on the number of ingredient occurrences in recipes. And then, preselected ingredient sets and positive examples are randomly sampled from each category set to have the almost same ratio, and negative examples are randomly sampled from all candidates. We adopt this process since it is expected that low frequency ingredients are basically more difficult than high frequency ingredients and we also need to evaluate this tendency. In the end, We prepare 4,536 tasks by preselected ingredient set size, where each task consists of 100 pairs.

As can be seen in Table 2, our models usually outperform NMF on one-to-one, two-to-one, and three-to-one test sets and it appears our models are considerably stable against the number of preselected ingredients. In order to clarify the effectiveness of the Relation Encoder module, we have added W/O RELENC model which is same as our models without the Relation Encoder module. W/O RELENC is similar performance with NMF on only one-to-one and horribly on two-to-one and three-to-one test sets in MAP@10. Apparently, the Relation Encoder module plays an important role in our models. For more detailed analysis, we compare Roc-curves for our models by ingredient frequency-based pair types(Figrure 4), which consists of High Frequency-to-High Frequency(H-H), High Frequency-to-Low Frequency(H-L), Low Frequency-to-High Frequency(L-H), and Low Frequency-to-Low Frequency(L-L), e.g. H-L describes pairs that a preselected ingredient is used in various recipes frequently, a paired ingredient is used in only specific recipes. Here, we report the result of Explicit model since Implicit model is also same tendency. As expected, L-L seems to be the most difficult task for our models, and H-L is follow. This result is some good since our model does not overfit to a low frequency ingredients overall.

Interpretablity At first, we visualize attention weights in the Explicit model for some specific food pairs. Figure 5 shows while *egg* is paired with *chocolate* for food categories or culinary conveniences, *miso* is considerably for flavor molecules related affinity, too. The result of *egg* is consistent with the results reported by De Clercq et al. (2016). Next, we focus on the attention weights of flavor molecules in our trained Explicit model for the quantitative analysis. We compare the average of Flavor Molecule attention weights with correlation coefficients which are calculated on full recipes. Details are in the caption of Table 3. In the ranking of attention weights, up to the top 5 are illegally high scores and listed ingredients does not seem to be related with flavor correlation. It would appear that since the number of ingredient occurrences is very low, the relational representations are not learned properly. On the other hand, the top 6 and below looks fairly good results. In the rank-

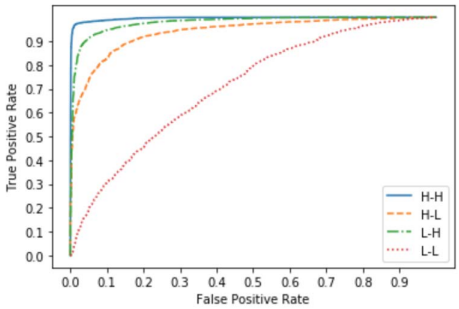


Figure 4: Comparison of Roc-curve by ingredient frequency-based pair type, which use the Explicit model on CulinaryDB, one-to-one pairs. L describes Low frequency, H describes High frequency.

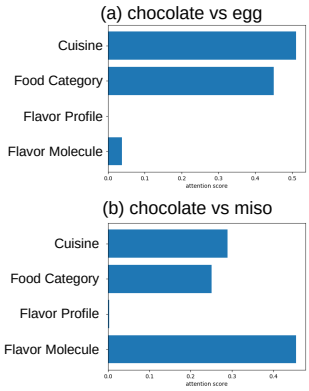


Figure 5: Visualizations of attention weights of our Explicit model on CulidnaryDB.

Table 3: Comparison of food ingredients sorted in descending order of scores related with correlation for flavor molecules. Left: ranked based on the average of Flavor Molecule attention weights in our Explicit model. Right: ranked based on correlation coefficients between the number of food ingredient co-occurrence in recipes and the shared flavor molecules number between two food ingredients. Both scores are calculated on the co-occurred ingredient pairs in recipes. Note that while the range of scores for our attentions weights is $[0, 1]$, the range of scores for correlation coefficients is $[-1, 1]$ and Freq is the number of ingredient occurrences in whole recipes.

Rank	Our attention weights			Correlation coefficients		
	Name	Score	Freq	Name	Score	Freq
1	soy yogurt	1.000	2	true frog	0.730	4
2	wheaten bread	1.000	4	florida pompano	0.577	3
3	sandalwood	1.000	2	shellfish	0.510	7
4	common dab	1.000	2	oil palm	0.488	3
5	blackberry brandy	1.000	3	abalone	0.408	4
6	miso	0.773	95	orange oil	0.403	8
7	wine	0.618	299	multigrain bread	0.358	8
8	sesame	0.603	1394	potato bread	0.356	7
9	gelatin	0.583	238	waffle	0.352	13
10	sherry	0.578	475	fruit juice	0.344	21

ing of correlation coefficients, there are some ingredients like some *breads* and *waffle* that seems to have little to do with flavor molecules. It looks like our models is able to find hidden ingredients structures within recipes different from simple correlation coefficients.

5 Conclusion

For ingredient recommendation systems, we proposed models, which rank ingredients based on co-occurrence of ingredients in cooking recipes and learn the relational representations which can give reasons as to why ingredients are recommended. In our experiments, we found that our explicitly/implicitly integrated factors can improve predictions, and our trained relational representation detected some correlations between ingredients and factors such as molecular structure. We hope these analysis encourage the discovery of new ingredient combinations discovery and new recipe creation. Some opportunities for future work are controlling ranking scores based on the manual modification of relational representations. In case some chefs want to know good combinations in terms of flavor, this functions will be helpful.

REFERENCES

Y. Ahn, S. Ahnert, J. Bagrow, and A. Barabási. Flavor network and the principles of food pairing. *Sci Rep 1, 196*, 2011. URL https://doi.org/10.1038/srep00196.

Ganesh Bagler. Culinarydb. 2017. URL http://cosylab.iiitd.edu.in/culinarydb.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26*, pp. 2787–2795. Curran Associates, Inc., 2013. URL http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf.

Marlies De Clercq, Michiel Stock, Bernard De Baets, and Willem Waegeman. Data-driven recipe completion using machine learning methods. *Trends in Food Science & Technology*, 49:1–13, 03 2016. doi: 10.1016/j.tifs.2015.11.010.

Neelansh Garg, Apuroop Sethupathy, Rudraksh Tuwani, Rakhi NK, Shubham Dokania, Arvind Iyer, Ayushi Gupta, Shubhra Agrawal, Navjot Singh, Shubham Shukla, Kriti Kathuria, Rahul Badhwar, Rakesh Kanji, Anupam Jain, Avneet Kaur, Rashmi Nagpal, and Ganesh Bagler. FlavorDB: a database of flavor molecules. *Nucleic Acids Research*, 46(D1):D1210–D1216, 10 2017. ISSN 0305-1048. doi: 10.1093/nar/gkx957. URL https://doi.org/10.1093/nar/gkx957.

A. Hermans, Lucas Beyer, and B. Leibe. In defense of the triplet loss for person re-identification. *ArXiv*, abs/1703.07737, 2017.

Balázs Hidasi, Alexandros Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. *CoRR*, abs/1511.06939, 2016.

Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y. Chang. Improving sequential recommendation with knowledge-enhanced memory networks. *In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pp. 505–514.ACM, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1400–1409, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1147. URL https://www.aclweb.org/anthology/D16-1147.

Donghyeon Park, Keonwoo Kim, Yonggyu Park, Jungwoon Shin, and Jaewoo Kang. Kitchenette: Predicting and ranking food ingredient pairings using siamese neural network. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 5930–5936. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/822. URL https://doi.org/10.24963/ijcai.2019/822.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks, 2015.

Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Latent relational metric learning via memory-based attention for collaborative ranking. *In Proceedings of the 2018 World Wide Web Conference (WWW '18). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland*, pp. 729–739, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 31*, pp. 5165–5175. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7763-link-prediction-based-on-graph-neural-networks.pdf.

A Appendix

A.1 Ingredient Set Encoder

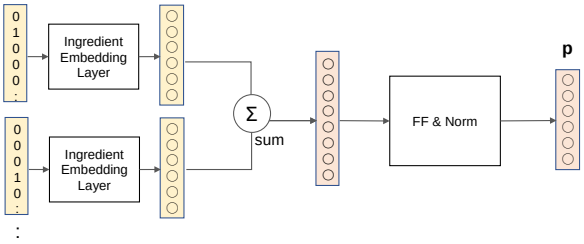


Figure 6: Ingredient Set Encoder

A.2 Triplets for Ingredients

ingredient	attribute	attribute value
apple	category	fruit
apple	flavor molecule	(+)-Delta-Cadinene
apple	flavor molecule	(-)-Epicatechin
apple	color	red
apricot	flavor molecule	(+)-Delta-Cadinene
apricot	category	fruit
:	:	:

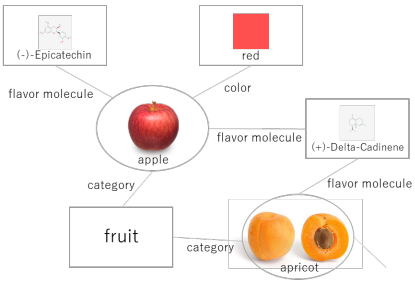


Figure 7: Triplets for Ingredients

A.3 Dataset Statistics

Table 4: Dataset Statistics

Datasets	Recipe Num	Ing./Molecule Num	Attribute Types
CulinaryDB	45,772 (train+valid: 41,231, test: 4,541)	658/1,788	cuisine category, food category, flavor molecules, flavor profile(e.g. Bitter)
Flavornet	55,001 (train+valid: 49,501, test: 5,500)	381/1,021	cuisine category, food category, flavor molecules

A.4 EXAMPLE RANKING RESULTS ON THE CULINARYDB

Table 5: Example Ranking Results on the CulinaryDB

Rank	Apple	Chocolate	Apple and Chocolate
1	canola oil	almond	canola oil
2	honey	banana	orange
3	mint	orange	almond
4	orange	cream	honey
5	cinnamon	raisin	raisin
6	white bread	cream cheese	mint
7	nutmeg	syrup	cream
8	lime	mint	corn
9	rice	canola oil	lemon juice
10	corn	corn	cinnamon