Inferring Ingredient Relationships From Recipes

Christopher Dinh
cdinh2@umbc.edu

Abstract

One of the primary skills of cooking is knowledge of what ingredients complement each other, which is largely learned by experimentation and experience. While there are lists of ingredients that contain this information, they are usually compiled by directly surveying professional chefs. That expert knowlege is useful, but I would like to see if similar resources can be created computationally. I intend to use Kaggle's Recipe Ingredients dataset for several experiments using various approaches to determine relationships between ingredients. In the first experiment, I will attempt to learn embeddings for ingredients in order to see what ingredients are used in similar ways. Next, I will attempt to use those embeddings to create a model that predicts what ingredients are most likely to be used with a given list of ingredients.

1 Problem

Learning what ingredients go together intuitively requires expensive and time-consuming experimentation to gain the experience required. There are resources like The Flavor Bible (Page and Dornenburg, 2009) which distill this experience by interviewing professional chefs, but those resources have several issues that are solved by a computational method for determining ingredient relationships. Chefs have personal preferences that would be reflected in them that may not be representative of overall trends. Also, the interviewing process needed to collect a significant number of chefs' opinions is much more costly than a computational method would be.

2 Solution

I used the Kaggle Epicurious - Recipes with Rating and Nutrition dataset (Kaggle, 2016) to train a model to recognize the relationships between ingredients. First, I used several methods described at (NSS, 2017) to create vector representations of the ingredients. Then, I used those vector representations as input for a MaxEnt model as well as a neural model that each attempted to determine what ingredient is most likely to result from a given recipe.

The models at (NSS, 2017) are designed for word embeddings, so I adapted them slightly to work with ingredients because there is no order. Rather than iterating over an ordered list of words that form a document, the context of an ingredient is equivalent to the recipe without that ingredient included.

3 Previous Work

The existing work in the realm of ingredient analysis that I have found uses Mutual Information to form a network of ingredients indicating which tend to occur together(Teng et al., 2012). However, that research primarily focused on using ingredients as features for a model that predicts the rating of a recipe rather than intending to see their relationships directly. In addition, it uses user comments to determine what ingredients can be substituted for each other rather than looking for ingredients that appeared in similar contexts.

The Continuous Bag-of-Words model that I used to create one set of word embeddings is described in (Mikolov et al., 2013) and it uses a neural network with a single hidden layer to find word embeddings. I modified it slightly to compensate for the lack of meaningful order in the ingredient lists. Rather than creating context vectors from some predefined number of words around the

word being predicted, every ingredient other than the one being predicted is considered part of the context.

4 Experimental Methodology

For the Epicurious dataset (Kaggle, 2016), I have manually categorized the features so that all non-ingredient ones are removed. Then, I split the data into 80% training, 10% development, and 10% test data sets.

4.1 Notation

Let V be the size of the list of ingredients I.

$$I = I_1, I_2, ..., I_V$$

Each recipe $R_i \subseteq I$ is represented by a vector $r_i \in \mathbb{R}^V$ such that

$$r_i = [r_i^{(1)}, r_i^{(2)}, ..., r_i^{(V)}]$$

$$r_i^{(j)} \in 0, 1$$

$$r_i^{(j)} = 0 \Leftrightarrow I_j \notin R$$

$$r_i^{(j)} = 1 \Leftrightarrow I_j \in R$$

Using these recipe definitions, a dataset $\Delta \subset \mathcal{P}(I)$ is defined as a tuple of recipes where N is the number of recipes in the dataset.

$$\Delta = (R_1, R_2, ..., R_N)$$

This is represented with the matrix $D \in \mathbb{R}^{N \times V}$

$$D = \begin{bmatrix} r_1^{(1)} & \dots & r_1^{(V)} \\ \vdots & \ddots & \vdots \\ r_i^{(1)} & \dots & r_i^{(V)} \end{bmatrix}$$

4.2 Co-Occurrence Matrix Embeddings

Using the training data, I created four vector representations of the ingredients.

The co-occurrence ingredient embedding matrix $C \in \mathbb{R}^{V \times V}$ is defined as:

$$count(I_x)$$

is the number of recipes ingredient number x appears in.

$$C = (D)^T \times D - \begin{bmatrix} count(I_1) & & 0 \\ & \ddots & \\ 0 & & count(I_V) \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & \cdots & C_{1,V} \\ \vdots & \ddots & \vdots \\ C_{V,1} & \cdots & 0 \end{bmatrix}$$

$C_{a,b}$ is the number of times that ingredients a and b occurred in the same recipe.

4.3 PCA-Reduced Co-Occurrence Matrix Embeddings

After a PCA dimensional reduction to some size $p < V$, the resulting ingredient embedding matrix $P \in \mathbb{R}^{p \times V}$ is

$$P = \begin{bmatrix} P_{1,1} & \cdots & P_{1,V} \\ \vdots & \ddots & \vdots \\ P_{p,1} & \cdots & P_{p,V} \end{bmatrix}$$

4.4 CBOW and Encoding Embeddings

The final two ingredient embedding matrices are determined using CBOW (Mikolov et al., 2013). To get embeddings of size n, a neural model is defined such that given a set of λ input vectors $x_1, x_2, ..., x_\lambda \ni \forall k, x_k \in \mathbb{R}^V$,

$$W_1 \in \mathbb{R}^{n-1 \times V}, B_1 \in \mathbb{R}^n$$

$$W_2 \in \mathbb{R}^{V \times n-1}, B_1 \in \mathbb{R}^V$$

$$\begin{Bmatrix} p(I_1) \\ \vdots \\ p(I_V) \end{Bmatrix} = softmax\left(W_2 \bullet \left(\frac{1}{\lambda}\sum_{k=1}^{\lambda}(W_1 \bullet x_k + B_1\right) + B_2\right)$$

Gradient descent with the ADAM optimizer is used to optimize these parameters to minimize negative log-likelihood. Then, the CBOW embedding matrix is $\Psi = [W_2, B_2]^T \in \mathbb{R}^{n \times V}$. I noticed that the "encoding" layer also has a shape compatible with the embeddings, so let the Encoding embedding matrix be $\Phi = W_1 \in \mathbb{R}^{n-1 \times V}$.

4.5 Evaluating Embeddings

Given any of these ingredient embedding matrices $E \in \mathbb{R}^{m \times V}$ such that

$$E = \begin{bmatrix} E_{1,1} & \cdots & E_{1,V} \\ \vdots & \ddots & \vdots \\ E_{m,1} & \cdots & E_{m,V} \end{bmatrix}$$

The vector representation of I_x is

$$I_x = [E_{1,x}, E_{2,x}, ..., E_{V,x}]$$

To translate a data matrix D into its embedding form $D' \in \mathbb{R}^{N \times m}$,

$$D' = D \bullet E^T$$

These embedding matrices were then compared by converting the training and development and test data into their embedding forms and training two models on them. The effectiveness of the trained models was used as a proxy for the usefulness of each embedding type.

I used a multinomial MaxEnt model which attempted to classify which ingredient went with a recipe given the recipe's embedding form. The model's performance was evaluated as its Exact Match accuracy, or the percentage of the test data that it classified exactly correctly.

The second evaluation model is a neural model with one hidden layer that was evaluated based on the average cosine similarity of its output to the embedding of the correct ingredient. The model took the form

$$W_3 \times max(0, W_2 \times max(0, W_1 \times x + B_1) + B_2) + B_3$$

In other terms, it had two hidden layers with ReLU activations and the output was a linear layer. It was trained with Mean Squared Error loss rather than cosine similarity because

As a more subjective measurement, I chose an ingredient and found the five closest ingredients to it in each embedding space as measured by cosine similarity.

5 Results

After preprocessing, there were 312 ingredients used with 14936 recipes in the training set, 1867 in the development set, and 1868 in the test set.

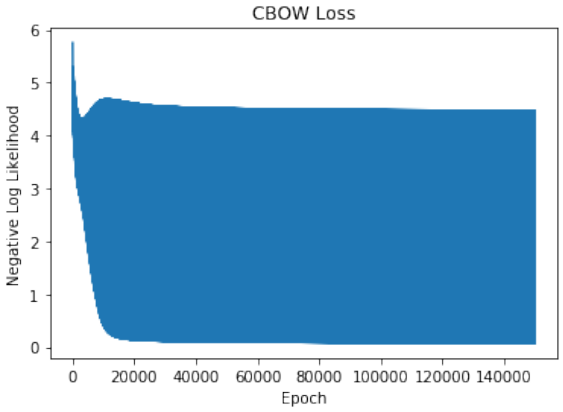
I tried several different sizes of embedding for PCA and evaluating them gave the results in Table 1.

Embedding Size	Exact Match Accuracy
10	8.03%
25	11.20%
50	13.04%
100	13.73%
200	13.91%

Table 1: PCA Sizes

Based on that table, I decided that a size of 100 was a good balance between performance and compressing the recipe vectors to a smaller size.

I used CUDA to train the CBOW model for 10,000 epochs, which still did not successfully classify any recipe.



Once all of the embedding matrices were created, I used the MaxEnt and neural models to evaluate them. To tune the number of epochs for the neural model, I ran it with each set of embeddings until the test loss started to increase and then set that point as the number of epochs to run for. Each of the figures is captioned with the number of epochs that I chose for it.

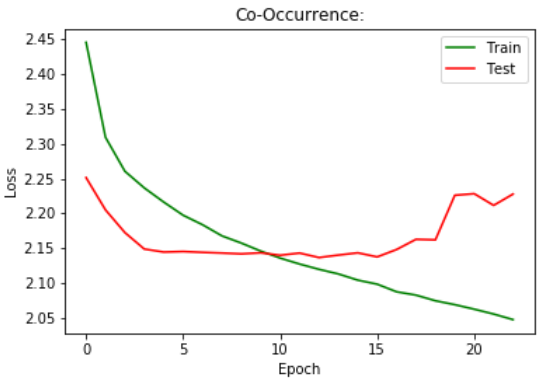


Figure 1: $n = 13$

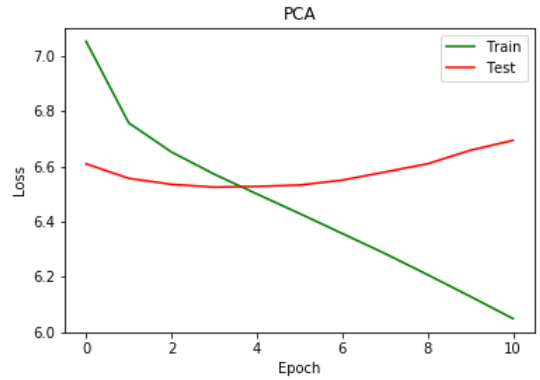


Figure 2: $n = 4$

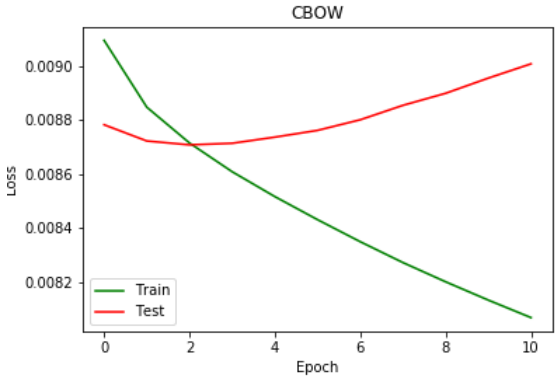


Figure 3: $n = 3$

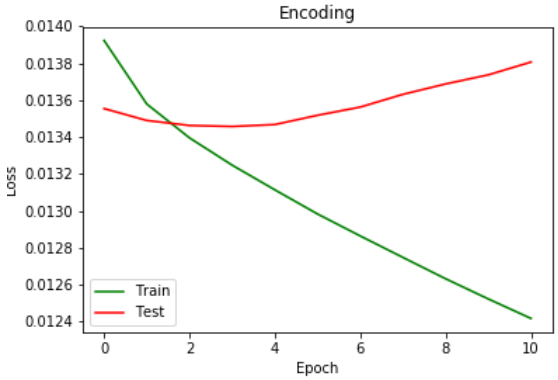


Figure 4: $n = 4$

Table 2 contains the results of my evaluations. MaxEnt Accuracy is the percent of recipes that were classified correctly by the MaxEnt model and NN Similarity is the average cosine similarity of the neural model's output to the embedding of the correct ingredient.

Based on these results, I concluded that my dataset was not large enough to effectively learn embeddings with a CBOW model, but it could

MaxEnt Model Results	
Embedding Type	**Accuracy**
Co-Occurrence	13.845%
PCA	13.530%
CBOW	14.476%
Encoding	13.858%

Table 2: Embedding Performance

Neural Model Results		
Embedding Type	**Accuracy**	**Similarity**
Co-Occurrence	9.492%	0.752
PCA	11.698%	0.408
CBOW	12.366%	0.238
Encoding	11.889%	0.223

Table 3: Embedding Performance

reach and even exceed the performance of the simpler co-occurrence based methods. Interestingly, while the classification accuracy was slightly higher for the CBOW-based embeddings, the average cosine similarity was lower, which I interpret as the CBOW models tending to spread the data out more, decreasing the average similarity overall.

I also found the five closest ingredients to "avocado" in each embedding space based on their cosine similarity.

Co-Occurrence Matrix		**PCA**	
Ingredient	Similarity	Ingredient	Similarity
jalapeo	0.810	jalapeo	0.585
bell pepper	0.792	hot pepper	0.469
hot pepper	0.776	corn	0.459
pepper	0.771	cilantro	0.422
corn	0.758	cucumber	0.421

Table 4: Co-Occurrence Avocado Similarity

CBOW		**Encoding**	
Ingredient	Similarity	Ingredient	Similarity
sour cream	0.570	jcama	0.769
mayonnaise	0.476	tortillas	0.526
jcama	0.468	sour cream	0.503
coriander	0.388	tomatillo	0.471
tortillas	0.354	soy	0.381

Table 5: CBOW Avocado Similarity

This shows that the CBOW embeddings appear to have learned some meaningful semantics even

from this small dataset such as the mexican cuisine that jcama, sour cream, and tortillas tend to appear in along with avocados.

6 Limitations

The dataset that I am using only has about 20,000 recipes in it, which is small enough that it makes training a neural network difficult for complex data like flavor interactions. Additionally, it did not include information like primary flavor molecules, texture, or other measurable characteristics that may be important latent variables in the creation of a recipe.

Finally, my model failed to achieve usable classification accuracies with the highest reaching 14.47%, which is not high enough for any practical application.

7 Future Work

I excluded many features that are present in the dataset which include information such as rating, nutrition, meal (lunch, dinner, etc.), and cuisine, so it could be interesting to include them in some experiments. Another potential source of additional features is the characteristics of the ingredients listed in (Page and Dornenburg, 2009) as those include some of the potential latent variables previously mentioned such as categorizing based on the primary tastes of sweet, sour, bitter, salty, and umami.

As a result of these data limitations, my model did not reach a useful classification accuracy, but because the CBOW embeddings were able to surpass the baseline embeddings created using the co-occurrence matrix, those models would likely perform better with more data, which would need to be collected and annotated.

In the long-term, with sufficient data, this concept could be used to create novel recipes and gain a better understanding of what goes into making a meal that tastes good.

References

Kaggle. 2016. Epicurious - recipes with rating and nutrition. https://www.kaggle.com/hugodarwood/epirecipes. Accessed: 2018-11-16.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

NSS. 2017. An intuitive understanding of word embeddings: From count vectors to word2vec. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/. Accessed: 2018-11-16.

Karen Page and Andrew Dornenburg. 2009. *The Flavor Bible*. Little, Brown and Company.

Chun-Yuen Teng, Yu-Ru Lin, and Lada A. Adamic. 2012. Recipe recommendation using ingredient networks. In *Proceedings of the 4th Annual ACM Web Science Conference*, WebSci '12, pages 298–307, New York, NY, USA. ACM.