Missing Ingredient Prediction

Courtney Moran
Electrical Engineering M.S.
Stanford University
moranc@stanford.edu

Masha Karelina
Biophysics Program
Stanford University
mashka@stanford.edu

Abstract

In this project we aim to train a model that can predict missing ingredients of a recipe given a partial list of the ingredients. Different approaches to representing the ingredients and architectures will be evaluated. Including one hot encodings, word embeddings, a simple neural network, and more advanced architectures.

Our repository can be found here: https://github.com/maschka/ingredients_predictor

1 Introduction

There are a lot of underlying patterns in recipes that allow humans to really develop intuition for recipes and the ingredients in them. This project aims at identifying if this pattern can exist in purely the ingredients alone. We want to predict missing ingredients of a recipe. That is, given a partial list of ingredients, can we train a model to suggest additional ingredients that would fit with that recipe. This is in essence a supervised learning project where the input is a list of ingredients where one item is masked or removed, and the label that is used to evaluate the loss is the removed item or items from the recipe. This means that this is actually a task that a lot of training data can be produced given just a few examples.
This may be an interesting and fun tool to use when trying to identify what items to include in a meal. One could input the contents of one's fridge to get inspiration for what to buy on the next grocery run. Additionally, because this project has a lot of overlap with NLP, this project could have some interesting extensions.

2 Related work

There has been a growing amount of research in the development of machine learning and food. A recent review highlights the importance that food has in daily life and that developing machine learning models in this field. Most of these methods are focused on image recognition and identifying the food in an image[3]. The focus of these applications is frequently on health and on improving production quality.

However, this project is focused on food in terms of flavor profiles. Intuitively we believe that recipes should have common motifs and that there are commonalities in the ingredients that could be learn able. Instead of focusing our attention on image recognition side of food, we instead treat food as an application for natural language processing methods.

There are project that have been tangential to our project. There are projects that have successfully attempted to classify cuisine from ingredients[4]. Additionally other projects have attempted to predict recipe names from food images[5].

,

Some of the insipiration for this project comes for the pre-training methods used on BERT and the masked language problem[6].

3 Dataset and Features

The dataset was taken from Recipe Box[1] which contains approximately 125,000 recipes taken from various food websites. These recipes span a large variety of cuisines and dish types. Each recipe includes a title, an ingredient list, and a list of instructions. For this investigation, we are only using the ingredients lists.

To pre-process the data, first the quantities were removed from elements in the ingredient lists. This includes both numeric values, and units (eg. cups, teaspoons, etc.). Strings from advertisements were also removed. See code for hardcoded list of strings we removed. Next, to clean the data a little more, descriptor words such as diced, chopped, minced, etc. were removed. To do this, we imported an English dictionary and used it to remove words that weren't nouns from the ingredients. This reduced the number of unique ingredients which created more overlap amongst training examples. After cleaning, there were approximately 17,000 unique ingredients in the dataset.

4 Methods

4.1 Vectorization approaches

We tested our methods against several different representations of the ingredients list. An initial approach was to split the ingredients up by line - so that each line represents one token. That representation was then one-hot encoded into a vector that we ran through a model. To represent the data, a vocabulary list was formed of every unique ingredient seen in the dataset. The recipes were stored as vectors using a one-hot encoding. In this encoding, the vector was all zeros except for at the indices corresponding to the ingredients included in the recipe.

The next step was creating labels, by masking one ingredient in each recipe. This was done by simply setting one of the ones in the data vector to zero, and creating a label which has a one only at the index corresponding to this mask.

The next approach was to use Google's Word2Vec[2] tool to generate vector representations of every word seen in the ingredients. For this, a pretrained dictionary of word to embedding vector mappings was loaded. This model was trained on Google News, and outputs 300-element embedding vectors. To represent an ingredient, we used the sum of the embeddings of the words in it. Then to represent a recipe, the sum of its ingredient embeddings were used (excluding the ingredient held out as "missing"). The label for a recipe was defined as the embedding representation of the "missing" ingredient. More concretely:

$$x_i^{(r)} = \sum_k w_{i,k}^{(r)}$$
$$x^{(r)} = \sum_{i \neq j} x_i^{(r)}$$
$$y^{(r)} = x_j^{(r)}$$

$w_{i,k}^{(r)}$ = embedding of the k^{th} word in the i^{th} ingredient in the r^{th} recipe

$x_i^{(r)}$ = embedding of the i^{th} ingredient in recipe r

$x^{(r)}$ = embedding of the r^{th} receipe

$y^{(r)}$ = label of the r^{th} receipe

j = index of the "missing" ingredient

A 70-30 train-validation split was used to divide the data.

4.2 Architectures

As a baseline we used a simple neural network that has two hidden layers. The first layer has 120 nodes, and then is followed by a ReLU activation function. There is a softmax function and a cross entropy loss at the output, since the labels are very similar to a classification problem.

For the model that used the word embeddings, we used a 5 layer neural network with ReLu activations. The hidden layers all had a dimension of 200. An Adam optimizer with a learning rate of 0.001 was used. The data was divided into minibatches, of size 2000. These hyperparameters seemed to work the best after tuning.

4.3 Clustering

While analyzing the data and investigating low accuracy we also did some initial analysis of the clusters formed by the recipes. Using the one hot encoding described earlier, we cluster the data using k-means clustering, using 6 clusters. We categorize cluster centers by determining what the cluster centers's indecides 95% of largest values are and backcompute them to get ingredients, and visually analyze them with word clouds.

5 Experiments/Results/Discussion

In our baseline model, a learning rate of 0.01 was used. The mini-batch size was 100. At first a smaller batch size was used, but this caused the model to train very slowly. With the chosen hyperparameters, the model trained well after 20 epochs.

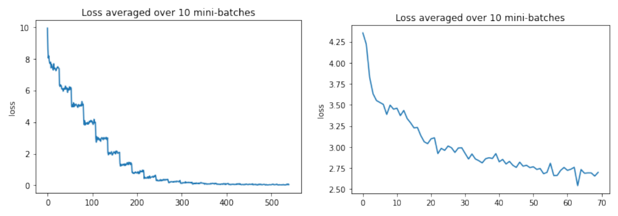

Figure 1: Training loss differences in first base model and deeper base model

The training and validation accuracies were calculated by computing the ratio of correctly predicted ingredients. The training accuracy was 99% while the validation accuracy was only 6%. Clearly there is an extreme issue with variance going on.

To impove high variance issues on the base model, we made several changes that resulted in better generalizability. We cleaned out our one hot encoding such that we only included tokens that had a frequency of at least 1%. We also attempted to make a more general dictionary by throwing out any word that did not contain a defintion that was a 'Noun'.With this much smaller dictionary, we had a testing accuracy of 17% which is an improvemnt over 6%. Additionally training accuracy resembled our testing predictions, also assuring us that we were not overfitting the data.

For the model with word embeddings, training converged after about 500 epochs. Computing accuracy in this case was a bit less obvious, since our output was not binary. Instead, we constructed a matrix of embeddings for every ingredient seen in the dataset, then mapped the indices of this matrix to the ingredients which generated them. The mapping of a model predictions was then chosen to be the index of the embedding with the smallest L2 distance to it. Once these mappings were obtained, accuracy could then be computed as the number of correctly predicted ingredients over the number of samples. The computed training accuracy was 24.4%, whereas the validation accuracy was only 6.4%. This was for exactly predicted ingredients. We also wanted a heuristic for determining how often the model predicted ingredients that were ’close’ to the correct one.

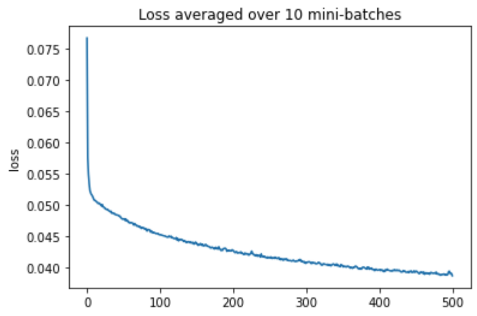

Figure 2: Training loss vs epoch of model with word embeddings

As a metric for ’closeness’ of predicted ingredients, we looked at the L2 norm of the difference between the predicted ingredient and the correct one. These ingredients were embedded with Google’s Word2Vec, which is trained such that similar words should have a small L2 distance, so this metric should make sense. Next, for some range of thresholds, we computed accuracies such that ingredients within the threshold distance from the actual one are counted as correct. The plots below show these accuracy heuristics vs threshold. Upon some exploration of the embeddings, it seems that predictions that are one word off, but similar, for example ’brown sugar’ vs ’white sugar’, tend to have a distance between about 2.7-3.5. In the plots, there is an evident jump in accuracy in the range of thresholds. If we consider these guesses to be ’close’, then the training set outputs ’close’ predictions $55 - 60\%$ of the time, and about $20 - 30\%$ of the time for the validation set.

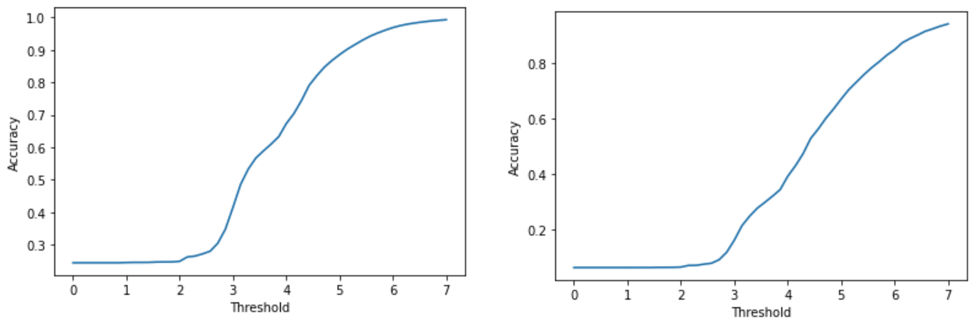

Figure 3: Training Accuracy Heuristic vs Threshold and Validation Accuracy Heuristic vs Threshold

5.1 Clustering

In the word clouds we see that the clustering of recipes is a little bit murky. This may be because in reality recipes vary a lot, and although intuitively we think we have a solid notion of ingredients, without much data refinement these recipes don’t lend to very easy targets.

6 Conclusion/Future Work

The baseline model did not generalize well, if at all. After looking at the data closely, the reason for this issue was that the there was not enough overlap of ingredients in the vocabulary. This is because

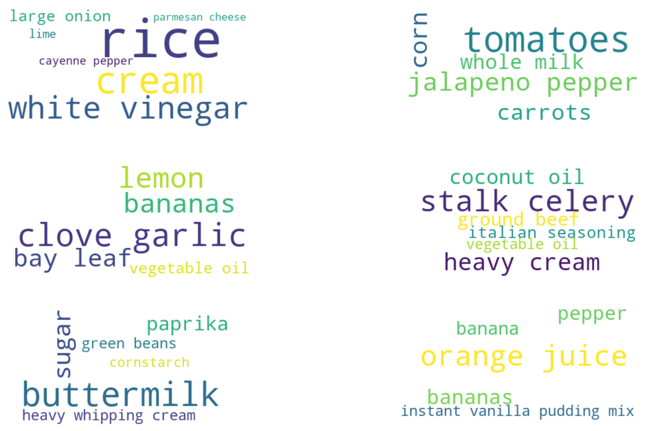

Figure 4: Word clusters generated from one hot encodings of recipes

the ingredients were mapped as exact strings, so while one recipe may call for "fresh parsley", and another for simply "parsley", these are mapped to different indices in the vocabulary, even though they are the same ingredient. Many ingredients include descriptors such as "chopped", "diced", "minced", etc, which made it difficult to find overlap of ingredients across different recipes. Because of this, we see that a direct mapping of ingredient strings is not sufficient for generalization.

To address this issue of generalization, we used Google's Word2Vec pretrained model on Google News. The embeddings were able to extract more descriptive features of the ingredients, and also provided a metric for measuring 'closeness' of ingredients. Overall, even with embeddings, the network didn't generalize as well as we hoped. Part of this was due to noise in the dataset. Many ingredients included words that were pushed together or not in the dictionary. Ultimately it seems that our embedding method for ingredients could use some improvements to boost performance. Simply taking the sum of the word embeddings in an ingredient and then summing over ingredients in the recipe introduces irregularites in the training data because not all ingredients have the same number of words, and not all recipes have the same number of ingredients. To address this issue, it would make sense to train a model that embeds an entire ingredient, not just summing over words. This would give more descriptive embeddings because sometimes words have different meanings when they appear together.

7 Contributions

Courtney - Data clean up, running training, embeddings
Masha - One hot encoding, setting up NN architectures, word clouds, clustering
Both - write ups, discussions, analysis, training, hyperparamater tuning.

References

[1] Lee, R. (2017, March 14). Recipe box. Retrieved February 27, 2021, from https://eightportions.com/datasets/Recipes/

[2] Google code archive - long-term storage for Google code project hosting. (n.d.). Retrieved February 28, 2021, from https://code.google.com/archive/p/word2vec/

[3]Zhou, L., Zhang, C., Liu, F., Qiu, Z. and He, Y. (2019), Application of Deep Learning in Food: A Review. Comprehensive Reviews in Food Science and Food Safety, 18: 1793-1811. https://doi.org/10.1111/1541-4337.12492

[4]Salvador, Amaia Drozdzal, Michal Giró-i-Nieto, Xavier Romero, Adriana. (2019). Inverse Cooking: Recipe Generation From Food Images. 10445-10454. 10.1109/CVPR.2019.01070.

[5]Paultimothymooney. (2018, June 30). Predict cuisine type from recipe ingredients. Retrieved March 19, 2021, from https://www.kaggle.com/paultimothymooney/predict-cuisine-type-from-recipe-ingredients

[6]Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.