A Recipe for Success

INFO 254 Final Report

By: Samantha Carr, Lia Chin-Purcell, Kailin Koch, Siqi Wu

Introduction

Cooking at home is an essential part of life and is becoming one of the challenges that people experience during Covid. With limited ingredients in my fridge, what can I make, and can I try something new? I crave some fancy dishes, but can I substitute the expensive ingredients to something affordable? What dishes can I make within 30 minutes? We wanted to use machine learning to answer these questions and approach recipes in new ways. With the recipe data, we tried to examine the following research questions: can we provide dish ideas to meet requirements, such as time constraints, kid friendliness, or ease of making? Given a set of ingredients, can we suggest substitutes that are cheaper, healthier or more environmentally friendly and can we generate a recipe automatically?

These questions are important for home cooks who want to find new recipes and ingredient substitutions to cook the food while still meeting their specific preferences and needs. The current Covid situation makes these questions imperative as people are forced to cook at home more often and deal with the reduced grocery shopping opportunities. New and innovative ideas of dishes to cook, possibilities to reduce food waste, as well as affordability become important measures to recipes.

Unfortunately, current recipe platforms can't provide satisfying solutions. Despite millions of recipes being available online, we found it difficult to search for alternative recipes and ingredients given constraints. Specifically, we looked at Food.com, a website with over 500,000 recipes. When searching for a recipe, Food.com provides basic filters such as "less than 30 minutes", "easy" and "main dish". But it fails to identify inexpensive or healthy alternatives. What's more, we searched for "chicken tomatoes" pretending we only have these two ingredients at hand, the resulting recipes include many more additional ingredients rendering themselves unusable.

In our project, we use a string of available ingredients, dishes and tags as input, and convert them to a vector representation. We use a word2vec to generate alternative ingredients and recipe names. We also use a recurrent neural network to generate a full recipe including instructions. We hypothesized that our word2vec model should be able to detect ingredients of the same category that are closer to each other (e,g. chicken is more similar to turkey than to coffee), and provides alternative ingredients based on constraints (e.g. the environmental friendly substitute of olive oil is another type of vegetable oil).

The Dataset

We found our dataset for this project on Kaggle. There were many different food-oriented options we looked at, but we chose this one because of the size, diversity, and content. It consists of over 230,000 recipes, and over 700,000 reviews, all from Food.com throughout the past 18 years. It contains all different courses (breakfast, lunch, and dinner recipes), and a variety of cuisines, from Mexican dishes, classic American dishes, to Asian dishes. Not only were there dishes, but also sides, condiments, and desserts. Additionally, it contained multiple variations of similar dishes, such as varying hamburger recipes and pizza recipes, which we knew would be good for ingredient comparisons and learning different permutations of ingredients.

We reviewed all the files available from the whole dataset in Kaggle, which included both raw recipe data and raw reviews data, and narrowed in on the raw recipe CSV to start our data exploration. We chose to focus on this set specifically because after discussing potential project ideas, our research questions and use cases were revolving around the actual ingredients and their combinations. Since in previous labs we had already used a reviews dataset from Yelp, we decided to leave that one, and perhaps come back to it if any feature engineering on the recipe dataset could benefit from information in the reviews. We wanted to explore ingredient substitutions, recipe similarities, and recipe generation, all of which required the recipe CSV.

Within the raw recipes file, we had 231,637 rows and 12 columns. Each row of the table was a different recipe with 12 columns of information. First, there was a "recipe name", which was an unformatted string of text indicating the name of the recipe as uploaded to Food.com. Next we had the recipe "id", which was a unique integer value associated with each recipe. Followed by that is "minutes", an integer value indicating the amount of minutes required to make the recipe. Next is "contributor id", a unique integer identifier of the person who uploaded the recipe, and "submitted", the date uploaded given in YEAR-MO-DAY. Next is the "tags" column, which is a non-formatted string of qualities assigned to the recipe by the uploader to help with the faceted search within Food.com. These are things like "main-course", or "30-minutes-or-less". This data looks like it is a list, but had to be pre-processed into a list object because the value in the column was actually just a string. Next was a "nutrition" column, which give the numeric value for each of the following nutritional values per serving size: calories (#), total fat (PDV), sugar (PDV) , sodium (PDV) , protein (PDV) , saturated fat. This column was the same as "tags", in that it visually looked like a list, but was actually a string value that we had to process into a parsable list for use later on. Next was "n_steps", an integer value indicating the amount of steps required in the recipe.

Following this column was "steps", which is a string value outlining all steps to the recipe from start to finish. Similar to tags and nutrition, this value was a string disguised as a list. It required some preprocessing, as the string value contained residual extra characters from once being a list, such as trailing brackets and backslashes. Following "steps" was "description", a one to two line string value explanation of recipe origin, special cooking tips, or general suggestions. Next was "ingredients", which was one long string value outlining all ingredients in the recipe. Similar to previous columns, this string value visually looked like a list, but was actually a string with lingering extra backslashes and brackets. This value thus also required pre-processing to break apart the ingredients using the comma separation for use in any model. The last column in the dataframe was "n_ingredients", an integer value for the amount of ingredients in the recipe.

Introduction to Approaches

The first step in our project was exploring datasets and finding a problem-space to work with. Each team member independently researched various datasets from places such as Kaggle, SNAP, the general social survey, and OpenML. We then came together as a group to decide our top most interesting problem spaces and their corresponding datasets, and landed on the Recipe dataset. After we landed on our project idea, we decided to do some initial data exploration and model experimentation independently. We chose to work independently on our models for two main reasons. One: our goals for our project did not have clear, substantive steps that two people could tackle at once, rather they lended better to each person exploring the possibilities for that goal. Two: we each had our own curiosities and interests within the problem space. Below we will briefly elaborate on each team member's individual work. While each team member worked independently on their models, we also contributed collaboratively to each other's models through our weekly meetings.

Kailin worked mainly with the Word2Vec model, and incorporated tags in the model's input data. She also experimented with adding recipe steps into the input, as well as working with doc2vec to better capture the recipe itself rather than just individual ingredients. Finally, Kailin worked on quantifying the goodness of the models and comparing our results to what exists for users currently. She experimented with comparing our models against models trained on the google news corpus, creating a function to evaluate the similarity of known similar words.

Samantha tried out including steps, ingredients, and tags in the input of a Word2Vec model to experiment with the model's context of a recipe. She then pivoted to working mainly with a RNN model after her initial exploration with Word2Vec. She decided to do word-level predictions with a RNN to generate recipe steps, instead of the character level we worked with in class. To hone the output of the RNN, she assessed various lengths of seed text, and also tuned hyper-parameters and utilized methods such as drop-out.

Siqi initially worked with the Word2Vec model and incorporated the recipe title along with the ingredients. She used NLP methods to parse meaningful words from recipe titles and incorporated them into the model. She also experimented with using various nutritional factors included in the dataset to find specific alternatives. Additionally, she tried out using a RNN to generate recipes using character level prediction. Finally, using Lia's environmental, unhealthy, and high emissions foods, she distinguished ingredients and names to be able to search the model more efficiently.

Lia initially explored the output of a Word2Vec model using only ingredients, and assessed incorporating ingredient alternatives for unhealthy foods based on a recipe's caloric total. Next, she searched for datasets on expensive foods and the amount of greenhouse gas emissions corresponding to each food to find alternatives. Finally, she experimented with different ways of incorporating these datasets to find optimal alternatives for costly ingredients.

Individual Work

Individual Work - Kailin Koch

At the project outset I compiled possible datasets and research questions we could evaluate, including a dataset on flood likelihood and another on the success rate of kickstarter project ideas. We discussed these possibilities as a team as well as meeting with Professor Pardos to get his perspective.

Once we had aligned on the recipe dataset, I started the exploratory data analysis and feature engineering ahead of creating a word2vec model. First, I cleaned up the dataset by stripping punctuation and numbers. I left two word phrases (ie black pepper) together, but left a space rather than connecting them into one token (ie not black_pepper). Then I created the corpus, first composed only of recipe names and ingredients. In my second iteration I added in tag elements. The tag field was very numerous, and I found better results focusing on 3-4 basic tags for each recipe (easy/hard, fast/slow, healthy/unhealthy) rather keeping all of them. Initially to evaluate the performance of my model I selected ~5 comparisons and my hypothesized result, and saw how these results changed as I modified the features and tuned the hyperparameters.

Here I went to Akshara's Office Hours. We were concerned that our recipe dataset was fundamentally different from a book. In the previous book examples we'd used, the context of surrounding chapters might make sense. Given the recipe dataset had no inherent ordering, however, we were worried the model might not make meaningful discernments between a meatloaf recipe next to a cake recipe, for example. Akshara recommended focusing on the window size to address this issue, and also suggested trying to bring the recipe steps in the model.

Adjusting my window size so that it was smaller (and thus more likely capturing only elements within a given recipe) improved the performance of my model. Similarly, limiting words for frequency > 5 removed some extraneous results. Adding in the recipe steps muddled some of my results. Instead of getting back recipes and ingredients as expected, the model often found the most similarity with verbs and adjectives related to the steps. ‘

In this time I also tried moving from the gensim word2vec model to a doc2vec instead. This, I hoped, would better capture the self-contained nature of a recipe without being influenced by nearby recipes. Additionally, I hoped it would do a better job of differentiating meals (ie lasagna) from ingredients (ie lasagna noodles). Unfortunately, this led to much less intelligible results. After reading more about doc2vec[1], my hypothesis is this was not the best tool for our use case. It’s strength is better understanding the inherent word ordering within a text. However, our queries of recipes depended little on the exact ordering of the ingredients within a recipe. Instead, I returned to the word2vec skip gram model and instead added words like ‘meal’ and ‘dinner’ to my queries to direct the model toward dishes rather than ingredients.

Lastly, I spent a lot of time thinking about how to best evaluate these models. Given 3 of us had worked on similar models, I wanted to better understand how we could eventually select the “best” performing model. As a gut check we came up with 5-10 comparisons and additions to try and see what types of results we were getting (as seen on slides 7, 9 and 10). However, this felt insufficient to me given that it was relying on a handful of anecdotal examples. Next, we decided to test the model against an existing model to get a baseline. We decided to use the google news model, but if time allowed would also have tried the fast text model. This comparison proved very interesting. While the google news model performed okay on our most simple examples, the more complicated ideas such as a ‘environmentally friendly beef’ went over its head. It was in these cases that our models shined.

Eventually I decided to try and create something comparable to the question-words[2] evaluation to evaluate our models. I created 2 separate functions and txt files (txt files are included in the zip file, functions are in my code). My first pass through, I created a list of words we knew should be similar to one another (apple and banana, etc). The function took the total distance between all of these word pairs, and in theory the model with the lowest difference captured these relationships best. Interestingly, I found the difference was smaller for the google model rather than our own models, which we could see were better capturing our more nuanced relationships. The group hypothesized that given the greater diversity in the google news corpus, it was possible the food words were closer together than the ingredients in our corpus, which was entirely food related. Additionally, I think the initial similarities were too simple. As such, they didn’t properly capture the nuances where our models did better.

Next I tried a similar approach, but this time using pairs. The first pair of words were supposed to be closer than the second set (ie beef emissions turkey emissions / veal expensive chicken expensive). This function simply took the percentage of words where the first word pair

[1]Le, Quoc, and Mikolov, Tomas. “Distributed Representations of Sentences and Documents.” *Proceedings of the 31 St International Conference on Machine Learning, Beijing, China, 2014.* https://cs.stanford.edu/~quocle/paragraph_vector.pdf
[2] Mikolov, Tomas, Chen, Kai, Corrado, Greg and Dean, Jeffrey. Efficient Estimation of Word Representations in Vector Space. September 2013. Footnote #1 http://www.fit.vutbr.cz/~imikolov/rnnlm/word-test.v1.txt

(ie healthy- spinach) was closer than the second word pair (healthy- butter). Using the difference comparison, my model got 96% while google news got 77%. I am not fully satisfied with this approach. I worry the set of words I chose isn't representative of all the recipes and relationships we want this model to understand. Additionally, it was challenging to use it as the deciding factor in our models because we all took slightly different approaches and focus areas. However, I think with more refinement this would be a successful approach to use because it's more standardized and generalizable.

Individual Work - Samantha Carr

Our group started by deciding to all bring a couple ideas to the table for discussion, so that we could decide a direction to go with the project. I spent some time looking through various datasets in different industries, such as Food, Health, Hotel, and Travel. I focused on exploring options that had machine learning potential (i.e. questions that could be explored specifically through models we had learned in this course), and not fall into just data analytics. This helped guide my search. I found datasets from each that I thought had valid research questions and ML applications, including two different food/recipe datasets, an Airbnb dataset, and a Bay Area bike sharing dataset.

After our group reviewed everyone's potential datasets, there were some commonalities around Food, which led us to ultimately pick the recipe dataset. Once selected, we discussed research questions we could answer, and decided we were all most interested exploring more with the Skipgram or RNN models for this project. We broke off to do initial data exploration and pre-processing that would be required to see how we could leverage the data to answer research questions. I started by reading the whole CSV file of recipes into a dataframe, and looking through each column to see format, content, size, and value-add to potential project models. The richest data for these models was in the recipe steps, and the ingredients, both of which required some preprocessing to correct formatting. Initially I wanted to focus on the research questions around substitution, and alternate dish recommendations, so I started with the word2vec model. I was interested to see what kinds of logical groupings could be observed with dimensionality reduction, as well as cosine similarity of the word vectors learned.

I wanted to start broad giving as much context as possible to the model, so I converted the name, tags, steps, and ingredients into one "recipe" string, which was considered a single sentence for the word2vec training. From here I trained and re-trained many models, varying the window size from 5-50, and varying the word minimum from 2-5. I had a set of hypothesized similarities I was testing against, such as distance between butter and oil versus butter and bananas. I was using this to gauge both the actual closeness of each, and also the comparative closeness of the two. With the corpus being all four columns, all my similarities were relatively low (even on the models with 5 minimum word count). And even though the butter/oil similarity would be greater than butter/banana, they were still both low.

To try giving more specific context, I dropped the name and the tags from the sentences, and kept just steps and ingredients. I also figured that to get more relational context between ingredients in multiple recipes, it would be good to try upping the minimum count of words to get a more clear picture. As learned from the Skipgram lab, if I wanted to do dimensionality reduction, too many words in the vocab create too much noise in the visualization. Ultimately, after using min count 12, I had the most clear groups without losing context in my visualization. In the tsne visualization code provided by Professor Pardos in the Word2Vec lab assignment, i could clearly see clusters for seafood, citruses, meats, numbers/counts, pastas, etc.

From here we met up with the group to discuss initial findings, and we narrowed in on specific tasks for each group member. I was interested in also exploring the recipe-generation idea we had discussed, so I decided to try this out as a next step, since both Kailin and Lia were having success with their Word2Vec models, and wanted to continue with trying to incorporate the tags.

My goal when starting on the RNN model was to output new and never-before-seen recipes that are similar to the recipes it trained on, but with logical variation. I didn't want to just memorize the recipes and predict the correct ending. I started with the RNN model we learned in class, and processed my data into the required format. For this model I decided to only use the recipe steps themselves. However, I quickly realized that the character count of my corpus, 122,216,858, was much too large for this model, and training it was going to be either impossible given my computing power in Colab, or take way way too long.

I did some research online, and found that instead of character prediction it was also possible to do word-level prediction. So instead of prompting the model to generate the next 30 characters, I could prompt the model to generate the next 30 words. This seemed a much more feasible idea, and given my goal was to generate recipes, it seemed a better choice for my intention. Additionally, it would ensure I was at least getting real words to start with, and not have to deal with gibberish words being generated. I found this article [3]to be particularly helpful, which gave an overview of how the word-level models work.

The pre-processing steps for this method are very similar, however instead of tokenizing characters into unique numeric values, I needed to tokenize the words. I read in all my recipe steps from all recipes in the dataset as one massive txt file, and broke it into a list by spaces. I cleaned it by removing punctuation and lower-casing everything. From here, I generated numeric tokens for each word in the document. The sequence length for training I knew should be larger, since for such a small corpus in our lab we used 10 characters. I started with a sequence length of 50 words for training, as that seemed to be a norm for word-level RNNs. I didn't want to go too big because it may not train as well, and I didn't want to go too small to lose context and perhaps risk forgetting.

However, when it came time to transform the tokens of the target word from the sentences (the target array 'y') into onehots, the notebook was crashing everytime. With over a million tokens, I realized that I should focus on a subset of the data, in order to get through the preprocessing and train the model without crashing. So from here on, I used only the first 50,000 tokens (words) from my corpus. For the first model, I started with an LSTM with two layers using categorical crossentropy loss function and adam optimizer. I trained over 100 epochs, ending with 60% accuracy. This first round of training took ~5 hours to complete. Since the goal was to generate new recipes, and not output known/learned recipes, I think 60% accuracy was a good starting point. High accuracy in this case, maybe anything above 90%, would have been perhaps too similar to the memorized data.

Using this first model, I tried generating recipe predictions using different combinations of unknown, partially known, and fully known seed text as input, and generated different length outputs from the model. The recipes it created made the most sense when given a sequence it had trained on as seed text, which does make sense. In general though, this model did a fair job of sentence structure. Instructions, while not always logical, did make sense language-wise for the most part.

[3] Brownlee, Jason. "How to Develop a Word-Level Neural Language Model and Use it to Generate Text". MachineLearningMastery

I wanted to experiment more with different hyperparameters, so I tried adding a dropout layer. This decreased accuracy by quite a bit, which makes sense since this layer is literally excluding neurons. I tried .5 at first, which really tanked my accuracy, but then reduced down to .2, and was still seeing pretty low accuracy. Keeping the dropout layer in, I increased the size of the layers from 100 to 130, and also upped the training epochs to 150 from 100. This combination of hyperparameters was ending at around 45% accuracy. From here, I tried removing the dropout layer but keeping the epochs high and increasing the two LSTM layers to 150 neurons. This combination produced the highest accuracy of around 80%. When testing this model, I did notice slight improvements in the predictions. It did a better job incorporating words mentioned in the seed text in the output text, and could generate somewhat cohesive outputs given completely unknown seedtext.

Overall, the models did a really good job of mimicking the format of how recipes are written, and generating some combination of words that represented that structure. It was good at producing the correct order of events as well. Knowing that mixing/stirring come before baking/cooking and eating, for example. However, it still needed some tuning to better incorporate the actual ingredients mentioned in the seed text into the steps output. I think more tuning of hyperparameters could have helped with this.

Individual Work - Lia Chin-Purcell

At the beginning of the project, I explored various datasets both on kaggle and on SNAP that I thought we might use to explore an interesting problem space. Some of the datasets that I brought up with the team were using the general social survey dataset and public voting records. Ultimately, our team decided to work with the recipe dataset from Kaggle.

Initially looking at the recipe dataset and discussing possibilities with my group mates, I was particularly excited by the idea of recommending ingredient alternatives based on various criteria. Initially, we saw that using "healthy" as a criteria might show interesting findings, since the dataset itself had information of calories as well as other nutritional facts. Our group was interested in using a Word2Vec model to recommend more healthy alternatives, but we were also considering how the number of calories a recipe has is a very imprecise and often faulty metric for healthiness. We also considered how the very premise and idea of recommending "healthy" things can be harmful, and maybe we could explore other criteria for alternatives. One problem with the idea of healthy alternatives is the western construction (influenced by the meat and dairy industry[4]) of various food groups. Indigenous nutrition educator Valerie Segrest[5] mentions eating locally and eating seasonally. Along the same idea, I considered suggesting environmentally conscious alternatives or cheaper alternatives.

After brainstorming some recommendation criteria, I started the data cleaning and "sentence" creation process. I decided to try feeding whole ingredients into my model (winter_squash instead of winter and squash) to try to conserve as many of the individual ingredients as possible, so in the end all ingredients could be searched over for alternatives. After creating the groups of ingredients, I trained a Word2Vec model. After this, I performed TSNE dimensionality reduction to visualize the ingredients in a 2-Dimensional space. I could clearly see similar ingredients would usually be closer to one another in the 2D space from the visualization, and from using the most_similar() function from gensim I could see similar

[4] Karla- Mason, Galen and Shi, Rebecca. "The Food Pyramid & How Money Influences USDA Dietary Guidelines". GreenChoice

[5] Segrest, Valerie and Krohn, Elise. "Feeding 7 Generations: A Salish Cookbook". WeRNative

ingredients would usually be closer than dissimilar ingredients in the 16D space as well. After creating my base model, I started including criteria for alternatives.

The first criteria I implemented was healthier options based on overall caloric total. As touched on above, this was not a perfect solution, because the calories listed were for the entire recipe and were not ingredient specific, and calories were not a perfect measure for healthiness. I continued with this approach though because it was the only data that was included in the original dataset, so meaningful recommendations were more likely. Additionally, it's likely that recipe creators would be considering overall calories when creating a recipe. For example, when making cookies, if there's already a significant amount of butter, so it's more likely the recipe writer would add other unhealthy ingredients like sugar. By adding a "healthy" tag for recipes under a certain caloric amount and "unhealthy" for recipies above a certain caloric amount to the models input, I was able to recommend healthier alternatives. Additionally, I used k-means clustering to help determine if recommendations were good by seeing if the original ingredient and the alternative were in the same cluster.

With the success of the healthy criteria, I moved on to finding and incorporating a dataset that would allow me to recommend alternatives based on greenhouse gas emissions. I used a dataset[6] with basic foods and their emissions, broken down by various stages in their production and distribution. Similarly to how I used the "unhealthy" tag, I included a "bad_emissions" tag for recipes with one or more foods that was above a certain threshold of greenhouse gas emissions. With this input, the model could suggest environmentally friendly alternatives, but was less successful than the healthy criteria suggestions.

Finally, I tried the same method for the environmentally friendly options for cheaper options. Unlike the environmental criteria, finding a dataset for cost of ingredients was much more challenging. For one, ingredient costs vary widely across different grocery stores, and often ingredients don't directly correspond to grocery store items. Also, combining the names of ingredients with names in a grocery store, or even food names in a government dataset, would require manual input for each ingredient. Instead of finding a dataset then, I used several sources to compile a list of common expensive ingredients such as vanilla beans and pine nuts. If a recipe contained one of these ingredients, I included an "expensive" tag in the input. Similarly to the environmentally friendly alternatives, the model could suggest some cheaper options, but was less successful than the healthy criteria suggestions.

Individual Work – Siqi Wu

Our group started by brainstorming possible datasets to apply machine learning techniques we learned. I focused on entertaining datasets such as trending Spotify, YouTube video and Netflix statistics. During the process, I found Google's Dataset Search platform that allows users to search for a dataset by theme. Using this platform, I found the recipe data for our project.

After our initial discussion on what we wanted to do with this dataset, my goal was to generate a full recipe including recipe tags, ingredients and instruction based on an input of a list of ingredients. In addition, I hoped my model would identify ingredients with high calories and recommend low-calorie alternatives. I started from cleaning and visualizing the dataset. I filtered out recipes that take more than 400 minutes to cook, have over 40 steps and used over 20 ingredients because these recipes are too complicated and thus not practical nor functional for the majority of users.

[6] Richte, Hannah and Roser, Max. "Environmental impacts of food production". Our World in Data.

I decided to use a character level long short-term memory (LSTM) recurrent neural networks (RNN) to train on these recipes using TensorFlow. I hoped this model could learn the dependencies at the character level as well as learn the difference between ingredients, tags, and steps. According to a blog on KDNuggets[7], adding an icon to mark each section of the recipe could help the RNN learn the structure better. Next, I tokenized the characters into unique numeric values to arrive at the vocabularies. I also added a stop character "_" to indicate the end of a recipe. With the vocabulary, I converted the recipes from text to a vector of 2000 numbers. Finally, when training the model, I used 20 epochs with 20 steps per epoch. I had to break the total recipes into 128 batches to avoid Google Colab to crash. I was only able to run the RNN model once because it took six hours to execute. Given the inputs of mushroom and apple, the resulting recipes were a gibberish of characters that did not make sense. I wanted to fine tune the hyperparameters such as the number of memory cells, the length of sequence of token, length of training epochs. However, the time required for each round of training was daunting to me, so I ditched this experiment.

At our group meeting, Kailin and Lia reported that they were able to get some interesting results using word2vec model. Lia was able to recommend healthier ingredients based on total calories. I was inspired by her and decided to expand her model to recommend not only ingredients, but also dishes that could meet a wider range of criteria.

I started from deciding what criteria can be used as input. Considering the tag column, I decided to incorporate the popular tags. I built a sorted dictionary with key being unique tags and value being the number of appearances. I took the top 27 tags as criteria for users to search. These tags included time constraints, such as "60-minutes-or-less"and "30-minutes-or-less", meal type, such as "main-dish" and "desserts", health indicates, such as "low-carb", "low-cholesterol", and "low-protein", as well as special requirements, such as "kid-friendly" and "holiday-event".

Next, I wanted to include the name of the recipe into the training dataset. The recipe names are specific to the dish. I wanted to make them more generalizable. For example, I want the name of "a bit different breakfast pizza" to be "pizza" so that it could capture all kinds of pizza dishes. I used the NLP package to parse the names and only left the nouns in the name. If the parsed result is an empty string, I used the original name of the recipe. This method was not perfect. It produced inaccurate results such as "kitchen" from "all in the kitchen chili", and "winter style" from "Arriba baked winter squash Mexican style". Considering the large amounts of data, these inaccurate results should not have a significant impact. To help the model learn the word phrases as a unit, I concatenated the phrases into one token using the hyphen (ie milk-shake).

Lastly, I revisited the ingredients list. I thought if I could remove the common ingredients in all dishes, the word2vec model could detect synonymous words and learn word associations more easily. As such, I created a sorted ingredients dictionary with the key being unique ingredients and values being the frequency counts. Based on the dictionary, I removed the top 20 ingredients from the training corpus, including salt, sugar, water, garlic, pepper and mixed spice etc. In retrospect, this step failed to impact the model since the unique ingredients we have for the model are too big.

Now my training text corpus had three fields, names, ingredients and tags. I kept the minimum count to 1 and tried a window size of 20. The model produced sensible results. I also compared my results to the results generated by genism Google news model. The Google news

[7] Trekhleb, Oleksii. https://www.kdnuggets.com/2020/07/generating-cooking-recipes-using-tensorflow.html

model is good at producing similar words but not creating alternatives. When I asked the Google news model to produce an alternative to pizza without flour, the results I got were “pizzas" and “Domino pizzas".

As I experimented with my model, I realized it can’t tell the difference between a dish name and an ingredient. When I asked the model to produce a dish that is similar to burger but without meat, the model would show ingredients such as onion. To help the model learn the difference, I included a stop word. I added “name-”, “tag-”, and “ing-” in front of each segment (ie ing-pasta, name-pizza, tag-kid_friendly). Then, in the input, I indicate the model if I want a name or ingredients as a result. Now, if I ask the model to produce the dish name of kid friendly main dishes (“tag-main-dish” + “tag-kid-friendly” + “name-”), it will return me dishes such as “pizza_snack_cups”.

I also included Lia’s environmental tags into my model. We hypothesized that an environmentally friendly substitute of olive oil is a different vegetable oil. However, my model failed this test and produced seasonings that are not a type of oil, such as “fresh-thyme-springs”. I hypothesized that the failure was caused by associating the entire dish with olive oil in it with the bad emission tag, as opposed to tagging the ingredients directly. I have yet to come up with a method to tag ingredients only.

I thought about how the model could continue to improve for industry uses. First, the text corpus for training could be stemmed and lemmatized so that a singular ingredient and its plural form should be counted as one (ie egg vs eggs). Second, the names of dishes should be refined to a generalized form. Third, continue to fine tune the model and try different combinations of window sizes and minimum word count in the model. Currently, with limited computing power, data cleaning and modeling takes an hour to run. Which limited my experimentations.

Conclusions

Our models perform comparably to Google news in interpreting basic ingredients, recipes
Our word2vec models did a good job of interpreting basic ingredients and their relationship to other dishes. For example, they understand that pasta is most similar to types of pasta like tortellini, turkey broth is more similar to chicken broth than turkey, and chicken and potato go together while cookie does not go with these words. We can see by looking at the dimensionality reduction and visualization that the models generate meaningful clusters of food types. For example, one cluster contains many baking ingredients such as flour and canola oil, while another contains various fall ingredients such as acorn squash and potatoes.

Our models do a better job of offering dish and ingredient alternatives for nuanced use cases than the Google news model or google search.
Once we explore more complicated relationships, our model starts to generate more meaningful similarities than Google news. For example, when we look for the output of burger - meat, our models produce results like ‘wrap’, while google news offers ‘cheeseburger’. Similarly, a query for ground beef - unhealthy yields ‘ground turkey’, while Google news suggests ‘beef imports’. Our models do occasionally produce less helpful results. Usually, this takes the form of offering an ingredient in a dish rather than a recipe alternative.

We have further work to do in combining the strengths of our models and evaluating them
Each of our word2vec models had unique wins. For example, Lia focused extensively on the expensive and environmental feature vectors, while Siqi dove into how to best distinguish recipe

names from ingredients. Unifying the strengths of these individual models will help streamline our approach. However, given we'd all made different choices about which features to include and the various hyperparameters, it will require more work to thoughtfully combine the strengths into a single model. Our first attempt to combine these models produced reduced performance. Additionally, we have laid the foundation for some standardization in reviewing the model performance. With the creation of the word comparison functions, we can move away from a handful of individual comparisons and towards a more consistent and automated evaluation process. This will allow us to meaningfully measure whether we are improving upon the model as we work to integrate the three models together (see Kailin's individual work section for more details).

Our RNN model shows promise at creating logical steps and sequences of a recipe.
Our RNN model was able to output logical steps and follow recipe sentence patterns, and if given enough starting seed text, could start to incorporate existing ingredients mentioned into the following steps generated. However, because the RNN model was trained on recipe instructions, ingredient-only seed text wasn't able to generate cohesive or relevant instructions. As such, there would be more work to do to make this model fit our use case of helping home chefs create their own recipes based on ingredients they have in the house. Still, it is promising to see that the RNN model can effectively parse the components of a recipe and make intelligible recommendations on existing recipes. Even a simpler version that could give basic instructions on how to prepare an ingredient could be very valuable to a home chef trying to figure out what to do with a new ingredient.

Real-World Implications

While we had many successes with these models, there is more needed before these are ready for the real-world. These include creating an intuitive user experience, continuing to tune our model so that it consistently returns dishes rather than ingredients, making a plan for ongoing training, and critically evaluating any biases in our initial corpus. For the RNN model, we'd also need additional computing power to continue training.

User Interaction

Our use cases are quite common and apply to many users. However, most people are not accustomed to thinking about recipes in the way our word2vec model assesses them (ie lasagna - slow + fast). When we discussed this in office hours with Priyam, he noted that we might need a classification model to determine the type of user request, such as whether they wanted a healthy version of a recipe, environmentally friendly, etc. Upon discussing as a group further, we realized that most of these requests were ongoing considerations. For example, most people who want to cook in an environmentally friendly way want to do so in all of their cooking, not just a single meal. As such, we concluded that an app that allows users to create a food profile could be the best way of accomplishing this. A user could input their dietary preferences, meals they enjoy and other considerations such as environmentally-friendly or price conscious. Users can pull up the app for times when they want to try something new, or get suggestions for an alternative option given their preferences.

Creating such an app would require additional programming, and we'd likely need to make adjustments to our model so that it could train offline or have a smaller version running online. We may need to make additional optimizations as well, such as using negative sampling, to avoid overly expensive computations online.

Model Tuning
As mentioned elsewhere, we'd want to do additional work to combine our 3 word2vec and increase the accuracy of returning either a dish or an ingredient. This would require standardizing the evaluation process, which is already underway. However, prior to this being released this stage would be essential. We'd also need to monitor the accuracy over time and ensure we didn't see a decrease. While some of our misses are understandable when you're familiar with the mechanics of word2vec, they are utterly nonsensical to an end user. For example, it makes some sense than when evaluating burger - meat, the model once returned mayonnaise. However, few vegetarians would be enthused with this recommended alternative.

Ongoing Corpus Updates
Our current models use a static corpus. If we were launching this live, we'd want a plan of attack for how this corpus would be updated and respond to new information. Maybe there's an API for food.com that we could use to pull down the most recent recipes or something equivalent. We'd also want an automated way to tag the environmental and expensive features, which we did a relatively manual process.

Corpus Evaluation
One big concern with this dataset is the composition of the recipes included. The most common recipe tags by cuisine were 'American', 'North American' and 'European'. Further work is needed to ensure that the model works as expected for less common cuisine types in the corpus, and potentially find a more diverse corpus to ensure it's not solely working for American and European foods. Additionally, we'd want to make sure that the foods from other cultures are representative of the cuisine as a whole (not just common American versions of these dishes), and were named appropriately. This article[8] discusses the adverse impacts of anglicizing recipe names, a common trend online.

Additional Computing Power
The RNN model took hours to finish running, which made optimizations difficult to implement just because the sheer time to run the model was so long. To efficiently train and evaluate this model we'd need more computing power.

Final Thoughts
This project successfully demonstrated that we could use basic word2vec and RNN models to meet unaddressed needs of home chefs to create meals that met their own needs, whether those were time constraints, kid friendliness, or being health, environmental or price conscious. Our next steps to bring this to life involve creating an intuitive user experience, standardizing our model evaluation, productionizing the model and ensuring a representative dataset for training.

[8] Makalintal, Bettina. When It Comes to a Recipe, What's in a Name? Vice News, July 2020. https://www.vice.com/en/article/889vvv/when-it-comes-to-a-recipe-whats-in-a-name