Recipe Writer Project
CS221 Final Project Progress Report

Bruno De Martino, Alex Lin, Austin Ray

Project Update

After receiving feedback from our original project, we decided to change directions. We believe our new project is more within the scope of CS 221 and that it uses the tools learned in class more directly. Our new project is a recipe writer, which learns from a database of existing recipes in order to produce new ones in a semi-random way - which makes things interesting. Our goal is to make creative and exciting new recipes that add new ingredients in unexpected and delightful ways, but still taste good. To make this work, we modify many of the strategies that we've learned so that they do not in fact produce the optimal choice - deterministic searches would always produce the same output for a given input, so instead of finding the "best" recipe, we find many "good" recipes.

The user will be able to input a minimum and maximum calorie amount for the dish, and/or what type of protein they prefer (e.g. fish, pork, chicken, tofu), and/or a set of ingredients that must be in the recipe. The recipe writer then uses these constraints to come up with a new recipe. We have a parser that takes recipes from cookbooks and other APIs (see developer.yummly.com) and separates them into individual ingredients and instructions, information which is used by our learning algorithm. We also have a parser that collects nutritional information from the government National Nutrient Database from calls to their API (see http://ndb.nal.usda.gov/ndb/foods), which will be used for the ingredient quantity definition part of our algorithm.

Databases

Recipe (Ingredients) Database: After making API requests to Yummly, we create a list of all recipes and store it in JSON mode. The Yummly API gives us a JSON object with all information on the recipe (e.g.: developer.yummly.com/documentation/get-recipe-response-sample). With that object, we create our own Recipe Database object, which consists of a recipe name, a recipe id, a list of ingredients, and a list of ingredient quantities with the ingredients. It is useful to get the ingredients separately from their quantities so we can quickly query the government National Nutrient Database and get the nutritional value for such ingredient. See Figure 1 for example entry on this database.

Nutritional Database: For each recipe we get when creating the Recipe Database, we store the ingredients for such recipe, and search for the ingredient in the government National Nutrient Database (NND) API. The NND API returns a JSON object with all nutritional information on the given food item (e.g.: http://api.nal.usda.gov/ndb/search/?format=json&q=butter&sort=n&max=25&offset=0&api_key=DEMO_KEY). With that object, we create our own Nutritional Database object, which consists of an ingredient name, the ingredient id, the amount of calories in 100g of that item, and a list of other units the item is usually referred as (e.g. tablespoon, packet, slice, oz) and their conversion to 100g. See Figure 2 for example entry on this database.

Recipe (Instructions) Database: To get our training data for instructions, we have spent a considerable amount of time scraping and parsing "Martha Stewart's Living Cookbook". This allowed us to create a database with 1229 recipes, with both ingredients (and their amounts), and the instructions (average of 5 steps per recipe). See Figure 3 for example entry on this database.

Creating the multiple databases and connecting them was a big challenge to us. We ran into multiple problems in terms of the time it takes per API call (each recipe takes 2 Yummly API calls, and each ingredient takes 2 NND API calls), and API call limits (for instance, the NND API allows for only 1,000 API calls per hour). Because of that, we had to run our database creation program for multiple hours to get a decently sized database (~10000). In the future, we intend to implement threading to be able to reach 1M+ recipes.

Choosing Ingredients

The first step of our project involves choosing the ingredients to be used in the recipe. There are two options for generating the ingredients upon which all of the other ingredients depend. The user either inputs a set of ingredients that they want in the recipe, or the initial ingredient is randomly generated. From this stage, we use a bigram-like Markov model to determine the rest of the ingredients. Specifically, processing the dataset gives us a data structure that contains information for each ingredient about which other ingredients that ingredient has been seen the most with. For example, a recipe that calls for chicken, potatoes, and corn will generate vectors of **{ chicken: {corn: 1, potatoes: 1}, potatoes: {chicken: 1, corn: 1}, corn: {chicken: 1, potatoes: 1} }**. If the next recipe contained chicken, corn, and ketchup, we would modify our data structure to be the following: **{ chicken: {corn: 2, potatoes: 1, ketchup: 1}, potatoes: {chicken: 1, corn: 1}, corn: {chicken: 2, potatoes: 1}, ketchup: {corn: 1, chicken: 1} }**. The frequency map is updated in this way for each recipe in our dataset.

With our frequency map assembled, we can begin constructing new recipes. Given initial ingredients (either supplied by the user or randomly picked), our algorithm adds new ingredients to the recipe one by one. Ingredients that have appeared more frequently in recipes with the ingredients in the current recipe are more likely to be picked. How many ingredients are used in this comparison (i.e. 2-gram vs 3-gram vs 4-gram) is decided by the user right now, but the optimal number to produce unique recipes seems to be around 3. Our method of weighted random choices prevents recipes from looking too much like existing recipes in our dataset.

Defining Quantity of Chosen Ingredients

After generating the ingredients, we will determine in what amounts they will appear in in our custom recipe. In order to meet the nutritional requirements determined by the user, we approach this problem as a CSP. The variables in our CSP are (ingredient, type) tuples that are used by separate functions to determine the nutritional content of each of our ingredients - for example (sugar, grams). The domains of each variable are mapped between the amounts that have been previously encountered in our recipe data. For example, if "eggs" have been seen to have amounts of 3, 5, and 8 in existing recipes, the domain of (eggs, single) would be {3, 4, 5, 6, 7, 8}, Given a value from the domain (e.g. 3 eggs), we look up the associated nutritional values and save it in an auxiliary variable as a constraint for the search.

Each assignment returned now falls within the specified nutrient range. We take a weighted random pick of the possible assignments, and then use ICM to determine the local maximum best assignment for our ingredient and amount choices. The weights of our ICM correspond to how often the amount of each ingredient appears in the ingredient dataset. For example, if salt is seen twice with the amount "3 tbsps," and once with "1 tbsp," the weight of "3 tbsp" will be 0.66 and the weight of "1 tbsp" will be 0.33.

Recipe Instructions

Once we have our list of ingredients, we need to create a cohesive set of instructions on how to prepare the newly invented dish. We currently have a baseline method implemented to produce a set of instructions, and we are working on a more involved implementation for the final product. Our current implementation operates in several stages. First, we look at our ingredient list and compare it with our instruction corpus to see if there are any pre-existing instructions that contain only a subset of our ingredients and no other ingredients. We insert those instructions verbatim into our instruction set. We do this until each of our ingredients has appeared in exactly one instruction or we run out of instructions that include only subsets of our ingredients (whichever comes first). For the leftover ingredients, we use simple instructions that we've seen in our instruction set and attempt to insert our ingredients into them in place of the existing ingredients in those sentences. As an example, a good simple instruction might be "Serve with bread.". Here, "bread" is the ingredient, its "before word" is "with" and its "after word" is ".". These before/after words are what make this sentence "simple", as almost any ingredient can be substituted for "bread" and the sentence will still make sense.

Our proposed "more involved" instruction writer uses a weighted-random beam search. States in this search are represented by the tuple (first word in current sentence, last word in current sentence). The initial state of our search problem is (ingredient name, ingredient name). The goal is to build an instruction sentence using an ingredient as the seed. Successor states are the following: (word_i, previous last word) for all word_i that have been seen to the left of the previous first word, and (previous first word, word_j) for all word_j that have been seen to the right of the previous last word. The rewards for reaching each successor state are calculated using a bigram cost model, where the costs are generated from the frequencies in which words appear adjacent to each other in our instruction corpus (much like homework 3). In the preprocessing of our instructions, we have added extra "START" and "END" tokens that signal the beginning and ending of an instruction sentence. The goal state of this search is reached when the first element in the tuple is equal to START and the last element is equal to END. From the initial state, we expand out in both directions, building the string until one of these terminal signals is reached, and from that point expanding out only in the other direction until the goal state has been attained.

In addition, not all instructions are created equal, and the order of the instructions has a significant effect on how a recipe actually works. Rather than taking a randomization of the ingredients, we will place limits on the corpuses that our instruction writer draws on. Since the beginning and ending instructions generally have specific qualities, such as preparation and serving, we will build the first and last instructions using only a corpus of first/last instructions we have seen in the corpus. We have also begun implementing a linear classifier whose inputs will be instruction sentences written by our recipe writer

and whose outputs will be a score describing how far towards the beginning of the instruction list that instruction should go.

Metrics

Due to the nature of our project, our measures of success are more qualitative than quantitative. However, there are some ways that we could test the coherence of our outputs. We plan to compare the output instructions against a general English language corpus in order to test grammatical correctness, as well as review it by hand to check for syntactical errors.

The ultimate metric of success is how the product performs, and how coherent, interesting, and delicious the generated recipes are. In addition, the purpose of the end product is to produce something with a sense of randomness and unconventionality that can delight the user and still taste good. As a result of the subjective and creative output that our project is designed to have, an "eyeball test" is actually a reasonable one to use (confirmed by a 221 TA in office hours)

Implemented Version

Currently, our project is primarily based on probabilistic Markov models. These give us some reasonable outputs, with ingredient sets that seem to go well together. However, our model for outputting instructions is less reliable, producing nonsensical (e.g. Place the juice in one layer in the center of the cross."), as well as some normal outputs (e.g. Spread the hazelnuts in a single layer on a rimmed baking sheet). In addition, it does not account for relations between ingredients (i.e. putting things in an oven without ever turning it on). While this gives us a technically working and entertaining tool, it doesn't always have the most logical outputs.

Next Steps

We have two main portions of our project that we are seeking to improve: ingredient choice and instruction definition. Our next experiments to improve the quality of our ingredient choices are to use unsupervised learning techniques such as K-means clustering to uncover more meaningful relationships between ingredients. In terms of instruction writing, we are investigating methods to better understand the syntax of the instructions to pull out meaningful phrases and ideas. For example, chicken is often grilled, or cooked in the oven at 375 degrees. We are also looking into methods to extract common actions, such as mixing or sauteing. We hope that implementing these will increase the sensibility of our instruction outputs.

Initial Experimental Results

Below are some initial results from our program. They have **not** been edited:

Mesoamerican Oil Hazelnuts with Mushrooms (SERVES 4)
Ingredients:
Vegetable oil
1 pound ground pork
1 cup mirin
1 cups packed light brown sugar

2 cups finely ground toasted hazelnuts
1/4 cup whole wheat flour
2 tablespoons extra virgin olive oil
1 ounce dried oyster mushrooms
Romano
1 cup grated Parmesan cheese
4 scallions

Instructions:

1. Heat the Romano in a large roasting pan over high heat.
2. Add the mirin; cook 1 minute.
3. When the flour is combined, return to heat.
4. Whisk in the oil.
5. Coat lightly with sugar, and set aside.
6. Add the mushrooms; return to a boil.
7. Spread the hazelnuts in a single layer on a rimmed baking sheet.
8. Add the oil in a slow, steady stream, working the pestle until emulsified; work in the cheese.
9. Add the scallions; stir until combined.
10. Serve with the pork.

Adonic Knockwurst Onions with Peppers (SERVES 8)

Ingredients:

9 knockwurst
1/4 medium green cabbage
1/2 teaspoons cider vinegar
1 medium onion
1/2 small red cabbage
2 small red onions
1 garlic bulb
1 cup mayonnaise
1/4 cup water
1/4 cup finely chopped red bell peppers
1/2 cups low sodium chicken broth

Instructions:

1. Heat the broth in a large saute pan over medium heat.
2. Add the water.
3. Transfer the onions to a large bowl using a slotted spoon.
4. Stir in the vinegar.
5. Add the onion, and stir to coat.
6. Cover; cook, stirring occasionally, until the cabbage is very soft, about 1 hour.
7. Place the tops on the peppers.
8. Add the knockwurst; cook, stirring, 1 minute.
9. Beat the mayonnaise in the bowl of an electric mixer fitted with the paddle attachment until it is creamy.
10. Drain on paper towels; season with bulb.
11. Serve warm.

Ionian Sugar Mushrooms with Butter (SERVES 12)

Ingredients:

Confectioners sugar
1/2 cup packed dark brown sugar
1 cups wheat bran
1 cup plus 2 tablespoon ground cinnamon
3 tablespoons unsalted butter
1 cup white wine
3 tablespoons roughly chopped fresh flat leaf parsley
4 pounds assorted mushrooms
1 tablespoon fresh lemon juice
1 cup coarsely chopped mint
10 tablespoons mayonnaise

Instructions:

1. Place the juice in one layer in the center of the cross.
2. Finely chop the mushrooms.
3. Bring to a boil over medium-high heat, stirring until the sugar is dissolved.
4. Melt the butter in a medium stockpot over medium-high heat.
5. Sprinkle with parsley.
6. Stir in 1 tablespoon of the mint.
7. Discard the cinnamon.
8. Leaving the pan with the bran in the oven, remove several at a time.
9. Add the wine to the saucepan.
10. Continually coat both sides with mayonnaise.
11. Serve immediately.

Appendix

Figure 1:


```
"Homemade Crispy & Easy Chicken Strips": {
    "ingredientLines": [
        "500 gms boneless chicken breasts",
        "2 tbsp ginger garlic paste",
        "1 tbsp crushed red chili flakes",
        "1 tsp black pepper powder",
        "1 egg",
        "salt to taste",
        "3 tbsp refined flour/maida",
        "2 cup corn flakes, crushed",
        "oil for frying"
    ],
    "ingredients": [
        "boneless chicken breast",
        "garlic paste",
        "crushed red pepper flakes",
        "black pepper",
        "eggs",
        "salt",
        "flour",
        "corn flakes",
        "oil"
    ],
    "recipeId": "Homemade-Crispy-_-Easy-Chicken-Strips-1365442",
    "recipeName": "Homemade Crispy & Easy Chicken Strips"
```


Figure 2:


```
"ketchup": {
    "foodCalories": [
        "101",
        "kcal",
        [
            {
                "eqv": 17.0,
                "label": "tbsp",
                "qty": 1.0,
                "value": "17"
            },
            {
                "eqv": 9.0,
                "label": "packet",
                "qty": 1.0,
                "value": "9"
            },
            {
                "eqv": 240.0,
                "label": "cup",
                "qty": 1.0,
                "value": "242"
            }
        ]
    ],
    "ingredientId": "11935",
    "ingredientName": "ketchup"
```


Figure 3: Example “Martha Stewart’s Living Cookbook” Recipe (verbatim)

shrimp summer rolls

MAKES 8

1 pound (about 30) small shrimp,
peeled and deveined

8 9-inch-round rice papers

8 large red-leaf lettuce leaves,
cut in half lengthwise

1 medium carrot, peeled, cut in matchsticks

1 small daikon radish, peeled, cut
into matchsticks

1 red bell pepper, seeds and membranes
removed, cut in matchsticks

1 bunch fresh chives, ends trimmed

1/4 cup packed mint leaves
Soy Dipping Sauce (recipe follows)

1. Bring a large pot of water to a boil. Reduce to a simmer. Add the cleaned shrimp; poach until pink and cooked through, about 2 minutes. Slice the cooked shrimp in half lengthwise. Set aside.
2. Fill a pan (large enough to hold the rice paper) with hot water. Dampen a clean kitchen towel with water; spread it out on a clean surface. Dip 1 sheet rice paper in the hot water for 5 seconds; transfer to the dampened towel, and smooth out (the wrapper will still feel hard, but it will soften as it sits).
3. Place 4 shrimp halves in a row, cut side up, 2 inches from the bottom edge. Place 2 pieces of lettuce over the shrimp. Top with more shrimp halves, some carrots, daikon, pepper, chives, and mint.
4. Fold the bottom edge of the rice paper over the filling. Continue to roll tightly so the shrimp halves are enclosed but still showing through the rice paper.

5. Place the finished roll on a plate; cover with a damp paper towel. Continue until all the ingredients are used. Serve with dipping sauce.