Final Project: Chef Raimsey Documentation

Time spent

24 hours

Homework Group

Maanya Goenka, Yemi Shin, Nicole Binder, Sue He

Repl Link:

`https://replit.com/@carlcs322s01s21/Final-Project-carlcs322s01s21-2#main.sh`

Breakdown

Maanya worked on accounting for allergies and substitutes via a brief chatbot-like interaction with the user, as well as refining recipe title generation, adding recipe formatting functionality, working on the initial draft of the project write-up, and on the presentation slides and the demo video; Yemi worked on preprocessing the recipe file corpus and on creating a graphical user interface; Nicole worked on the scraping code, recipe generation, and summary generation using GPT-2; Sue worked on generating a name for the recipe as well as correctly categorizing the recipe based on its similarities to existing recipes using Doc2Vec. All team members worked on debugging and populating the corpus files.

CS 322 - Final Project: Chef Raimsey Documentation **Small Ears**

Project Name: Chef Raimsey

Goals: Chef Raimsey is a dessert recipe generator that draws from a corpus of multiple recipe text files scraped from AllRecipes.com, and generates a well formatted recipe of ingredients and corresponding proportions for the user using a fun graphic display. It takes the user's favorite ingredient in a dessert and the provided allergies into account when generating recipes.

NLP Techniques Used: Doc2Vec Vectorial Semantics Model, Web Scraping and Extraction, Unigram Custom Tagging, Tokenizing, Corpus Preprocessing, GPT-2 Text Generation.

Project Components: Here, we will provide a detailed breakdown of the project in terms of the important folders/files in the repl link to the project.

- *topiclists*: This folder contains five .txt files, namely Cakes.txt, Cobblers.txt, Pies.txt, Cookies.txt, and Frozen desserts.txt. Each file is populated with approximately 30 urls which link to recipes that fall under the corresponding category and are derived from https://www.allrecipes.com/.
- *corpus*: This folder contains five sub-folders titled Cake, Cookies, Cobblers, Pies, and Frozen desserts. These folders in turn contain text files where each text file is composed of three components - the recipe name, the recipe summary, and the list of ingredients and proportions of each ingredient.
- *testcorpus*: This folder is just a smaller version of the *corpus* folder, again with the same five sub-folders. Each sub-folder contains a few text files used solely for code testing purposes.
- *tagger_dicts*: This folder contains different text files that will be used to generate a dictionary to be passed into the Unigram Tagger as the model. Unigram Tagger's model is a dictionary mapping a key (which is a token) and a tag (i.e. ING - for ingredient). For example, in 'preprocessing.py' each line in 'ingredients.txt' will become a token key, and will be mapped to the tag 'ING'. Often, each line looks like: (ingredient name),(ingredient-name). The second item after the comma is a 'single-word' version of the previous item, and prior to other preprocessing, all instances of the first item will be replaced by the second item. This is to ensure that cases like 'sour cream' will not get tokenized to 'sour' and 'cream' separately but instead as a single word 'sour-cream.' In addition, cardinals representing the ingredient itself, for example, very specific ingredient names like '(16 ounces) package pie mix', will be treated as an ingredient '(16-ounces)-package-pie-mix' because 16 ounces is part of the description of the ingredient and not necessarily the cardinality of the ingredient. This is because we can have measures such as 2 (16-ounces) packages of pie mix and in this case '2' is the cardinality of the ingredient.
- *allergies*: This folder contains only one sub-folder called allergens.txt, which stores a mapping for a primary allergen and all possible dessert ingredients which could contain that allergen. It is referred to when dealing with allergies in the dessert recipe so as to eliminate any ingredients from the recipe that the user is allergic to.

- *newly_generated_recipe*: This folder contains five sub-folders namely Cakes, Cookies, Frozen desserts, Cobblers and Pies. These sub-folders contain a varying number of text files storing the final generated recipe from the NLP model. Each file is composed of three parts - the generated name of the recipe, the generated summary, and the generated ingredients and proportions of ingredients used for the recipe. The name of the recipe is the name of the file that stores the recipe as well.

- *scrape_recipes.py*: This file helps us scrape recipes from the allrecipes web pages using the latest version (bs4) of BeautifulSoup. BeautifulSoup is a Python library used for pulling data out of HTML and XML files. Here, bs4 uses HTML tags to pull only the relevant information from each page i.e. the recipe name, the brief summary that accompanies the recipe and the list of ingredients for the recipe. It then writes this information onto new text files stored under the corresponding category folder which in turn lies under the broader *corpus* folder.

- *test_tagging.py*: This program is just for testing different POS taggers. Unigram Tagger was ultimately chosen because it was customizable.

- *preprocessing.py*: This program is responsible for preparing the different mappings that will be used by Chef Raimsey (eg: dictionary-of-frequently-seen-together-ingredients) as well processing the corpus text files to create a list of Recipe objects.

  - *Recipe*: This is an object representing a single recipe. This class contains the name, summary, list of ingredients, as well as the type of the recipe.
  - create_tag_dict: This function uses the text files in tagger-dicts to create a dictionary mapping token to tag. This dictionary will be used in training the Unigram tagger.
  - preprocess: For each recipe file in the corpus we, 1) replace instances of multi-word single ingredients (i.e. sour cream) to a single word connected by underscores (see above for more detailed explanation) as well as replace instances of multi-word preparation (i.e. finely chopped) to its single-word counterpart, 2) remove unnecessary punctuation (i.e. commas), 3) for each ingredient line (i.e. '1 ounce of chocolate syrup') in the recipe, tag the ingredient line, 4) resolve ambiguities in the ingredient description (see below for more details), 5) update the four dictionaries that will be used by Chef Raimsey to generate the recipe (see below for more details), 6) create a recipe object and append it to the list of recipes, after all files have been processed, 7) train the doc2vec model on the list of recipes.
  - resolve_ambiguity: The ultimate goal of this function is to get the ingredient line to be in the form [CD][UNIT][PREP][ING]. The function takes in a single tagged ingredient line, and then does the following things: 1) in examples such as '1 cup and 2 teaspoons', we have [CD1][UNIT1][None][CD2][UNIT2]. In this case, we want to add the smaller unit (second cardinal) to the larger unit (first cardinal), thus creating the clause [CD1 + CD2][UNIT1] which will be resolved to a [CD][UNIT] tag. 2) in examples such as '1 to 2 tablespoons of sour cream', we have [CD1][None][CD2][UNIT]. In this case, we want to average CD1 and CD2 and get [avg(CD1,CD2)][UNIT] in our final tags. Generally, these

tagged items are processed in the order they appear in the ingredients line, and are each appended to a single list that they pertain to (i.e. CD items are appended to 'cardinal' list, UNIT items are appended to 'unit' list etc). If any of the list contains more items, we take the first item off that list as the representative token of that category. As for the ingredients variable, it is constantly updated as new ING tag is encountered. In other words, if there are multiple ingredients in a single line, the last one is chosen. As for prep, if there are multiple PREP tokens, they are joined by "," to form strings like "sliced, chopped". The return value of this function is a strict, length 4 array of the form [cardinal,unit,preparation,ingredient] where each slot is either a tagged token that was ultimately chosen for that category (i.e. ("strawberries",ING)) through the above processes, or an "" if the list for that category empty.

- update_frequently_seen_together_ingredients: This method takes in a completely processed and refined list of ingredients for a single recipe and updates the dictionary-of-frequently-seen-together-ingredients. dictionary-of-frequently-seen-together-ingredients maps each ingredient to a list of ingredients it appear together with across the entire corpus.
- update_frequently_seen_used_amount: This method takes in a completely processed and refined single line description of an ingredient and updates the dictionary-of-frequently-used-amount. dictionary-of-frequently-used-amount maps each ingredient to a list of cardinals that the ingredient is used for across the entire corpus.
- update_frequently_seen_used_unit: This method takes in a completely processed and refined single line description of an ingredient and updates the dictionary-of-frequently-used-unit. dictionary-of-frequently-used-amount maps each ingredient to a list of units that the ingredient is used for across the entire corpus.
- update_frequently_prepared_method: This method takes in a completely processed and refined single line description of an ingredient and updates the dictionary-of-frequently-used-method. dictionary-of-frequently-prepared-method maps each ingredient to a list of preparations that the ingredient frequently undergoes across the entire corpus.
- allergen: This method processes `allergen.txt` and creates a mapping between the category of allergen, and a list of ingredients that pertain to that allergen category.
- create_ingredients_corpus: From the list of recipes generated via preprocessing, creates a recipe-corpus, which is a list of recipe ingredient string (so instead of a list, it would be a string) and a list of recipe types. These lists are used to train doc2vec.
- train_doc2vec: This method takes in the list of recipes, creates the ingredients corpus, trains a doc2vec model on the corpus, and returns the model. Doc2Vec is used in categorizing the generated recipes and naming the recipes.

- *chef_raimsey.py*: This Python file is where the heart of the project lies. It has a class Chef_Raimsey with several instance variables including the user's favorite ingredient, the recipe name, the list of user allergies, etc. The principle methods associated with this class include:

  - find_frequently_paired_ingredient: uses a pre-filled dictionary to randomly choose from a list of frequently paired together ingredients when given an ingredient as a key.
  - find_frequently_used_amount: uses a pre-filled dictionary to randomly choose from a list of frequently paired together ingredient quantities when given an ingredient as a key.
  - find_frequently_used_unit: uses a pre-filled dictionary to randomly choose from a list of frequently paired together ingredient units when given an ingredient as a key.
  - find_frequently_used_prep: uses a pre-filled dictionary to randomly choose from a list of frequently paired together ingredient preparation methods when given one ingredient as a key.
  - num_of_ingredients: returns a random number between 6 and 12 that determines how many steps the recipe will include.
  - generate: makes use of the user's favorite food to kick off the recipe generation process. Returns a Recipe object comprising the name of the recipe, the summary of the recipe, a list of ingredients for the recipe and the type or category that the recipe falls under.
  - add_new_recipe: creates a new file in the newly_generated_recipe folder with the corresponding recipe information.
  - format_recipe: formats the recipe so as to account for missing elements in the recipe ingredients list, to account for the use of plural quantity measures when it is and isn't required, and to get rid of unnecessary whitespace and newline characters.
  - categorize: uses Doc2Vec vectorial semantics to find which category of desserts the generated recipe belongs to the most and then assigns it to the recipe of that type.
  - name_recipe: generates a well formatted name for the recipe which includes the user's name and is customized based on the list of ingredients that are part of the recipe. If a user is allergic to a specific ingredient, then the name of the recipe indicates explicitly that that allergen is not present in the recipe.
  - create_summary: uses InferKit and a variation of the Marcov chain model to construct a summary to accompany the dessert recipe.
  - conversation_starter: engages in a conversation with the user and extracts information about the user such as name, favorite food (if the user's favorite food or a variation of it is not present in the list of ingredients that we have access too, a random one of those is selected to kick off the recipe generation process.

- *chef_raimsey_graphic.py*: This program is responsible for generating a basic graphical user interface for Chef Raimsey. Uses Zelle's simple graphics module.

  - conversation_starter_graphic: A graphic version of chef_raimsey's conversation_starter method. Is responsible for collecting information such as User Name, Favorite Ingredient, Allergies etc to generate a recipe.
  - display_recipe_graphic: Method called by generate in chef_raimsey.py to display the recipe on the graphic window.

  - reset_display: Method to reset the display after the user has clicked 'Start Over' in chef_raimsey_controller.py

- *chef_raimsey_controller.py*: This program is responsible for being the Controller, or the middle person between chef_raimsey.py and chef_raimsey_graphic.py. This third entity is necessary to avoid circular import problems when the two aforementioned programs need to coordinate their action together.

  - start: Call Chef Raimsey's `generate` method. After the method ends, call the method `start_over` to potentially start the game over.
  - start_over: Creates two buttons on the window, Start Over, and Exit. If the user clicks Start Over, a window is reset using reset_display method of chef_raimsey_graphic and the `start` method is called again.