# Building a Recipe Search Engine

Kiyoon Jeong

## *1 Introduction*

There are various kinds of recipes in the world. Some recipes might have the same name, but use different ingredients. Some recipes use an oven, and some use a gas range. In terms of cooking directions, cooking time is also an important factor, and nutrition could also vary depending on the amount of ingredients used. These features are mostly considered when people choose a recipe. In my search engine, these factors would be used as a ranking score, or used as a filter. Additionally, people might want to search for a specific name of recipe, a type of recipe, or a similar recipe. In this case, it is better to use clustering on the recipes. It would also be considered when calculating a ranking score. Also, it is always important to reflect the user feedback on our scoring process. A few more features, such as average rating and the number of reviews would be considered in scoring.

It is also important to deal with the query. The users might search recipes with detail, for example, “Please give me a recipe that can be made without an oven and could be prepared in 1 hour.” , or “low fat and low sodium sandwich”. Thus, catching an intent of the search query should be implemented. After filtering and scoring, it would print out top 10 recipes.

## *2 Dataset*

The dataset is present on Kaggle produced by Elisa[1]. The dataset was constructed via collecting Allrecipe.com recipe data. Allrecipes.com is one of the biggest recipe websites where 1.5 billion users visit per year. The data is a collection of recipes posted between 2000 and 2018. For this project, the raw-data_recipe.csv dataset is used. 49698 recipes are present in total. Each recipe has the following columns : recipe_id , recipe_name, aver_rate, image_url, review_nums, ingredients, cooking_directions, nutritions, and reviews. The cooking direction column contains cooking time, and nutritions column contains the amount of following nutritions : calories, sugars, caloriesFromFat, calcium, sodium, folate, fiber, thiamin, magnesium, iron, potassium, saturatedFat, cholesterol, vitaminC, carbohydrates, fat, niacin, vitaminA, vitaminB6, protein.

| | recipe_id | recipe_name | aver_rate | image_url | review_nums | ingredients | cooking_directions | nutritions |
|---|---|---|---|---|---|---|---|---|
| 0 | 222388 | Homemade Bacon | 5.000000 | https://images.media-allrecipes.com/userphotos... | 3 | pork belly^smoked paprika^kosher salt^ground b... | {'directions': u'Prep\n5 m\nCook\n2 h 45 m\nRe... | {u'niacin': {u'hasCompleteData': False, u'name... |
| 1 | 240488 | Pork Loin, Apples, and Sauerkraut | 4.764706 | https://images.media-allrecipes.com/userphotos... | 29 | sauerkraut drained^Granny Smith apples sliced^... | {'directions': u'Prep\n15 m\nCook\n2 h 30 m\nR... | {u'niacin': {u'hasCompleteData': False, u'name... |
| 2 | 218939 | Foolproof Rosemary Chicken Wings | 4.571429 | https://images.media-allrecipes.com/userphotos... | 12 | chicken wings^sprigs rosemary^head garlic^oliv... | {'directions': u'Prep\n20 m\nCook\n40 m\nReady... | {u'niacin': {u'hasCompleteData': True, u'name'... |
| 3 | 87211 | Chicken Pesto Paninis | 4.625000 | https://images.media-allrecipes.com/userphotos... | 163 | focaccia bread quartered^prepared basil pesto^... | {'directions': u'Prep\n15 m\nCook\n5 m\nReady ... | {u'niacin': {u'hasCompleteData': True, u'name'... |
| 4 | 245714 | Potato Bacon Pizza | 4.500000 | https://images.media-allrecipes.com/userphotos... | 2 | red potatoes^strips bacon^Sauce:^heavy whippin... | {'directions': u'Prep\n20 m\nCook\n45 m\nReady... | {u'niacin': {u'hasCompleteData': True, u'name'... |

[Table1 : Original Data]

## *3 Data Preprocessing*

### (1) Feature Selection

First, the reviews would not be considered in this project. Instead, average rating and number of reviews would be used as a user feedback. Recipe_name, ingredients, cooking_directions columns are used for scoring, and only a few parts of nutritions are used for filtering. People mostly care about fat, sugars, calories, and sodium. Thus, I added these 4 as new columns in a dataframe. In general, people search low amounts of these nutritions, so declare the value as 'True' if the amount is less than the average amount, and as 'false' if not. The next step is dividing the cooking_directions columns into 'cooking time' and 'recipe' columns. The cooking time has three sections, preparation time, cooking time and total time ('Ready In'). People mostly don't separate the cooking time and preparation time, so I only considered total time. The cooking type is displayed as minutes, hours, or days. To simplify, hours and days are converted to minutes and stored as an int value.

### (2) Edit Columns

The columns with a numeric value, such as average rating, number of reviews, and cooking time, could be used directly when scoring, however the columns with a string value, such as recipe_names, ingredients, and recipes could not. Unlike the usual web search engine, tf-idf score, which is calculating frequency of word in document and frequency of document that contains that word, doesn't make any meaning. For example, mentioning 'potatoes' 100 times doesn't have a big impact compared to mentioning it once. Also, rare ingredients don't need to be scored high. Also, the scoring process would be held on the entire filtered dataset accessing each column's value. Thus, inverted index structure is not needed for this case. Instead, the 'set' object would be stored as a value, which contains words in that column. 'NLTK' library is used for stemming each word. First, 'recipe_name' is replaced with a set of words. Same for 'ingredients' column. The cooking tools are important as much as ingredients, and it

could be found in the cooking direction columns. The tools would be used in scoring with the ingredients at the same time, so I added tools on the 'ingredients' column and renamed it into 'ing_ tool'.

### (3) Additional Column

People usually search for a specific recipe or recipe that is similar to another recipe. However, it is hard to label them by hand because of the cost, and hard to define the categories. The best possible method could be using the 'Latent Dirichlet Allocation' method. [2] It is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. Recipe name, ingredients, and tools are chosen to generate topics. For this, I created an additional 'tokens' column that temporarily stores aggregation of those three columns data. 'Scikit-learn' library is used to tokenize this column. Before training, the number of topics should be considered. On average, there are 148 words in each element in the 'tokens' column. And about 100 recipes for each topic might be a good choice. Because there are about 50000 recipes in this dataset, I choose the number of topics as 500 at first. As a result, some topics contain less than 10 recipes. To avoid creating meaningless topics, I tried some lower number of topics. Overall, 100 topics were enough to avoid topic less than 10 recipes, and pretty well distributed among the topics. [Image 1] shows the list of the words for each topic. After training the LDA model, I transformed the tokenized column again to get the topic label. [Table2] is the final dataframe that would be used in the query search part. Recipe_name, ingredients, and recipe columns are used as an output of search results, therefore not dropped. Aver_rate, review_nums, time, name, and ing_tool columns are used for scoring, and sugars, fat, calories, sodium, and topic columns are used for filtering. The set of words in recipe_name, ingredients, and recipe is also stored along with the dataframe, which would be used in the query processing part.

```
Top 10 ingredients for topic 0
['sugar', 'bowl', 'powder', 'oven', 'egg', 'degree', 'flour', 'cup', 'baking', 'muffin']


Top 10 ingredients for topic 1
['pepper', 'simmer', 'minute', 'pot', 'heat', 'chicken', 'celery', 'broth', 'onion', 'soup']


Top 10 ingredients for topic 2
['powder', 'salt', 'oven', 'soda', 'egg', 'sugar', 'degree', 'pan', 'baking', 'flour']


Top 10 ingredients for topic 3
['green', 'stir', 'skillet', 'onion', 'cook', 'minute', 'heat', 'bell', 'sausage', 'pepper']
```

[ Image1 : LDA : Top10 ingredients for topic (only shown 4 topics)]

| | recipe_name | aver_rate | review_nums | ingredients | time | recipe | name | ing_tool | sugars | fat | calories | sodium | topic |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Homemade Bacon | 5.000000 | 3 | pork belly^smoked paprika^kosher salt^ground b... | 710 | Preheat oven to 200 degrees F (95 degrees C). ... | {bacon, homemade} | {the, black, pepper, let, until, ., with, heav... | True | False | True | False | 96 |
| 1 | Pork Loin, Apples, and Sauerkraut | 4.764706 | 29 | sauerkraut drained^Granny Smith apples sliced^... | 165 | Preheat oven to 325 degrees F (165 degrees C).... | {,, and, loin, sauerkraut, pork, apple} | {roast, least, brown, f, apple, at, 325, toget... | False | True | False | False | 9 |
| 2 | Foolproof Rosemary Chicken Wings | 4.571429 | 12 | chicken wings^sprigs rosemary^head garlic^oliv... | 60 | Preheat an oven to 350 degrees F (175 degrees ... | {wing, foolproof, rosemary, chicken} | {chicken, touching, least, from, brown, ', f, ... | True | False | False | False | 85 |
| 3 | Chicken Pesto Paninis | 4.625000 | 163 | focaccia bread quartered^prepared basil pesto^... | 20 | Preheat a panini grill. Slice each quarter of ... | {pesto, chicken, panini} | {forming, the, pepper, prepared, chicken, unti... | True | False | False | False | 27 |
| 4 | Potato Bacon Pizza | 4.500000 | 2 | red potatoes^strips bacon^Sauce:^heavy whippin... | 70 | Place potato slices in a deep skillet and cove... | {pizza, bacon, potato} | {creamy, form, you, heavy, from, softens, brow... | True | True | True | True | 84 |

[Table2 : Preprocessed Data]

## *4 Query Processing*

Unlike the other general search topics, recipe search could be categorized explicitly. Most common case is searching with a recipe name. Or people might search for a similar recipe. In case of running out of certain material, people might search for a recipe without ingredients, or without kitchen tools. Also, people might want to take care of their health so that searching for a recipe that is low fat, low sodium, or low calories. Time is also one of the most important factors to choose the recipe. People might want to find a recipe with special categories, such as nationality, season, or time (Breakfast / Lunch / Dinner). The special categories search needs the additional columns, and it is hard to classify and label them in terms of cost and time. Except for this special case, the dataset is enough to implement this search ability. However, the most challenging part is how to catch the intent of the search query. People may give a few words, such as a recipe name. Also, people may give a sentence, such as “Please give me a Korean recipe that is not spicy and could be done without an oven.” In this case, using all words in a query is very inefficient, and it might give a spicy and oven-required recipe. Thus, the query needed to be preprocessed before being used.

### (1) Query Preprocess

Firstly, what kind of words would be considered? The word classes are useful categories to deal with this problem. In the search process, nouns are the most important one. Also, the adjective and adverb are important for catching a feature of a recipe. However, some are relatively or absolutely not

important, such as verb, pronoun, determiner, or conjunction. Thus, it is better to exclude words with those tags first. In a code, after tokenizing the query with the same method we did in the preprocess step (if not, same words could be expressed in a different way), NLTK library's pos_tag function is used to get the words' tags. As I mentioned above, 'VERB', 'PRON', 'DET', and 'CONJ' tags are excluded.

Next step is catching a negative flag. People usually say 'without', 'no', or 'not' to express a negative. In a code, firstly it checks if the word is 'no' or 'not' and changes the negative flag into 'True'. Also, it checks if the element is 'without' in case of 'ADP' tag (adposition), and changes the negative flag into 'True'.

Time also needs to be checked. People usually mention the cooking time like 'in an hour', or 'within 10 minutes'. The strategy is that whenever it checks the 'NUM' element, store it in a time_set variable. This variable is initially set as 1 in case of 'an hour'. To catch the cooking time intents, time_list is used which contains 'h', 'm', 'hour', 'minute', and 'min'. If the word is in this time_list, then append the value of a time_set variable into the search queue list.

If the word is not in the above cases, it would be appended to the search queue list. However, in case that negative flag is true, append '!' in the front of the word to denote the negative.

```
sentence = "low fat, and does not contain nuts"
```

```
['low', 'fat', ',', 'and', 'doe', 'not', 'contain', 'nut']
```

```
[('low', 'ADJ'), ('fat', 'NOUN'), (',', '.'), ('and', 'CONJ'), ('doe', 'VERB'), ('not', 'ADV'), ('contain', 'VERB'), ('nut', 'NOUN')]
```

```
['low', 'fat', '!nut']
```

[ Image2 : Input query -> Tokenization and stemming -> Tagging -> Preprocess Result ]

### (2) Catching Intents

Word list is created. The next step is categorizing these words with the proper intents. In the data preprocessing part, I mentioned that nutrition, time, ingredients, tools, and topic would be considered in the scoring part. Thus, I created each corresponding intent list, and a negative intent list. In a code, first, check if the element type is integer. Previously, only the time was stored as an integer type, thus stored it in a time intent list. Then it checks if the word starts with '!'. '!' means it is negative, so put it on the negative intent list. In case that element is one of the nutritions, it stores the element in a nutrition intent list. Lastly, among the leftovers, if the word is in the token list, it would be appended

to the search word intent list. The token list is a previously created set which contains all the words appeared in recipe, recipe name, and ingredients.

The topic intent part is not as clear as others. The first strategy is predicting a topic with the query using a LDA model. If the query is large enough to find the right topic, it would be a good measure. However, people may input a single word which is too small to choose the right topic. The next strategy is finding a recipe which contains the words in query the most, and declaring the topic value as that recipe's topic. It would satisfy the user who searches for a specific recipe name. However, it would perform badly if the user put lots of words or sentences. I choose to use both strategies which could complement each drawback. 1 topic would be chosen based on query, and 10 topics would be chosen based on the recipe name score. In a code, because the count vector and LDA model are required, I trained those models first. After building that model, the first topic would be predicted using the query. The next topic basically computed after the recipe name scoring part. It chooses the topics from Top 10 score recipes. I choose relatively more topics on the second method because the name is the most important factor. Thus, if the query contains keywords that are the most important factor to choose a certain topic, then it would give a few topics.

### *5 Filtering and Scoring*

In the previous steps, I could improve the performance based on the result. For example, if the blacklist could be built which stores meaningless words, such as food, recipe, or delicious, then we could pool important keywords only. Also, creating more and more useful intents and building more sophisticated classification tools would probably improve the search results. However, this filtering and scoring part is somewhat intuitive. Because we could not get the user feedback, it is hard to tell whether the performance is good or bad. Thus, I decided the scores of each intents based on the importance. Importance was decided by my intuition. The overall procedure is following : 1. Get the scores from the recipe name. 2. Get top 10 score recipes' topic 3. Filter the dataset that the topic matches 4. Filter the dataset that the nutrition column is true. 5. Get the scores from the 'ingredients and tools' (ing_tool) column. 6. Get the scores from the 'average rate' and 'review number' columns. 7. Get top 10 score recipes.

(1) Filtering

Firstly, some intents could be used as a strict criteria, such as 'Topic' and 'Nutrition'. 'Topic' feature is originally built for checking relevance. Because there are 100 topics in total and we choose 10

topics among them, it would filter out 90% of data. People who care about nutrition, generally don't want to get the recipes that don't match their preference. We set half as a criteria of nutrition, so it would filter out 50% of data. To sum it up, about 5% of data would be used for the next step, which is about 2500.

(2) Scoring

I set the importance based on the situation analysis. First, people mostly search the name of the recipe, or some keywords that could be in a name, such as nation, key ingredients, or type of cuisine. Thus, 'Recipe name' score should be the highest. The next important one should be the negative intents. Even though it matches the name, that recipe should not get the high score if it contains unwanted tools or ingredients. The ingredients and tools' score is the next important one. Time is quite controversial. Some might think the time constraints should be strict, but I think that it is better to consider the recipe which needs bearable extra time. The last one is review and rating. Searching relevant recipes is the first, and getting good recipes is the next. Thus, these columns' scores would not change the result much compared to other columns.

In detail, name score is calculated as following : 10 * $matchwords^2$ / length of name. It could reflect the portion of matching words and also give more score on match number. For example, a recipe that has two match words out of three gets a higher score than a recipe that has one match word out of one. In case of matching a negative word, 4 points are deduced from the total score, and in case of matching an ingredient or tool, 2 points are added on the total score. Time score is used as a deduction factor. If the cooking time is less than the time constraint, then it would not deduct, and if not, it would deduct 1 point per 5 minutes. The number of reviews could be used as a measure to see how much this recipe is tried and check if the average rating is trustworthy. Thus, it is used to add an extra bonus on average rate. The maximum possible bonus is set as 20% of the average rate. The document score is calculated as the following : min(row['review_nums']/100,1). The recipes with more than 100 reviews would get a full bonus point on their rating. I set the maximum value of this score to 1. Thus, the adjusted score divided by 5 is added to the score. Because of the bonus, the maximum score is 1.2. Finally, it sums up all scores and returns Top 10 recipes.

## *6 Results and Performance*

Because it is training the LDA model, it takes 10 minutes for setup. It takes about 1~2 seconds to get a search result. 'Tabulate' library is used to print out the result table. Users can search recipes and select the recipe for details. Mostly, it prints out relevant recipes, but still has some issues. Because it

doesn’t catch the meaningless word, it might cause a problem. In image 2, the user query contains ‘food’ which results in getting the ‘Fairy Food’ or ‘Kwek Kwek’ (Fillipino Street Food). Thus, checking the possible meaningless words and building a blacklist could solve this issue. The other issue is the keyword of food. Currently, name intent and ingredients & tool intent use the same search words for scoring. In image 3, the user searches the seafood pasta that contains tomato and basil, but the Top4 recipes are all about tomato basil pasta, and ‘Cajun Seafood Pasta’ recipe is 5th place. However, separating ingredients words and recipe name words is nonsense. The possible way is checking which words could play a key role in recipe name and give extra bonus on them. This could be done by building a list that contains important keywords, which needs a lot of effort. The other possible way is calculating each word score using the document frequency. The rare word would get more points. However, rarity does not mean importance in recipe search. Thus, building a keyword list by hand might be more useful and effective in the long run. There is an issue in time scoring. Because some recipes don't mention cooking time, it is hard to determine whether to deduct score or not. The last issue is how many results should it print. I print out the top 10 results just for convenience. In practical terms, it is better to print out the result that is higher than the threshold value.

```
Recipe Search :Give me a Korean food which could be made without oven. It should be low fat and take less than a hour.
+-----------------------------------------------------------------+
| recipe_name                                                     |
|-----------------------------------------------------------------|
| Korean Spicy Chicken and Potato (Tak Toritang)                  |
| Chompchae Deopbap (Korean Spicy Tuna and Rice)                  |
| Kimbop (Korean Sushi)                                           |
| Korean Marinade                                                 |
| Fairy Food                                                      |
| Korean Squash                                                   |
| Korean Pizza                                                    |
| Quick and Simple Korean Doenjang Chigae (Bean Paste/Tofu Soup)  |
| Vegan Japchae Korean Noodles                                    |
| Kwek Kwek (Filipino Street Food)                                |
+-----------------------------------------------------------------+
Choose Recipe(1~10) :1
Ingredients :

 chicken drumettes
large potatoes
carrots
large onion
garlic
water
soy sauce
white sugar
gochujang (Korean hot pepper paste)

Cooking Time :  60  min

Cooking Direction :

-  In a large pot over medium heat, mix the chicken, potatoes, carrots, onion, sugar, and garlic.
-  Pour in water and soy sauce, and stir in sugar and hot pepper paste.
-  Bring to a boil, reduce heat to low, and simmer 45 minutes, until chicken juices run clear, vegetables are tender, a
-  Serve with hot cooked rice.
-
Back to list[1], Start new search[2], Stop[3] :
```

[Image 3 : Irrelevant Recipe]

```
Back to list[1], Start new search[2], Stop[3] :2
Recipe Search :seafood pasta which contains tomato and basil and could be cooked less than 30 mins.
+-----------------------------------------+
| recipe_name                             |
|-----------------------------------------|
| Tomato Basil Pasta                      |
| Tomato Basil Pasta Salad                |
| Tomato Basil Penne Pasta                |
| Homemade Tomato Basil Pasta Sauce       |
| Cajun Seafood Pasta                     |
| Fresh Tomato Pasta                      |
| Tomato Basil Spaghettini                |
| Tomato Basil Tagliatelle                |
| Easy Olive Oil, Tomato, and Basil Pasta |
| Tomato Basil Chicken                    |
+-----------------------------------------+
Choose Recipe(1~10) :1
Ingredients :

 diced tomatoes
small onion
olive oil
crushed garlic
leaves fresh basil
fusilli pasta
grated Parmesan cheese
crumbled feta cheese
salt and ground black pepper to taste

Cooking Time :  40  min

Cooking Direction :

-  Stir tomatoes, onion, olive oil, garlic, and basil together in a bowl.
-  Bring a large pot of lightly salted water to a boil.
-  Cook fusilli in the boiling water, stirring occasionally, until cooked through but firm to the bite, 12 minutes.
-  Drain.
-  Toss warm pasta with feta cheese and Parmesan cheese in a large bowl.
-  Stir tomato mixture into pasta and season with salt and pepper.
-
Back to list[1], Start new search[2], Stop[3] :
```

[Image 4 : Keyword]

## *7 Conclusion*

The recipe search engine is more straightforward than usual. Therefore, it needs less effort on finding the right result. However, query processing is very important to catch the users' search intent. It needs lots of effort to analyze the context and to pick the important words. Most of the part could be done by using a natural language processing model, such as 'Countvectorizer', 'Stemming' and 'Part Of Speech Tagging' method. Also, lots of study and experiments are needed to build the extra lists, such as the meaningless word list and important keyword list.

This dataset doesn't contain the category of recipes. If it has, then that category could also be used as one intent. Instead, I used the 'Latent Dirichlet Allocation' method as one of the categories. It is useful to filter out the irrelevant recipe.

Recipe websites could be improved a lot by utilizing the user's feedback. For example, A/B testing could help adjust scoring coefficient and check the importance or meaningless word. Also, we could analyze the user's query to update the keyword list. Recipe tends to have a trend in season or in region. These data could be used as one of the intents. User's review could also be helpful. We could

analyze the important factors that users care a lot about. Based on it, we could crawl, scap, or make better recipes.

## *8 Reference*

[1] Elisa, 'food RecSys-V1', Kaggle, 2018-09-14, 'https://www.kaggle.com/elisaxxygao/foodrecsysv1'

[2] Wikipedia, 'Latent Dirichlet Allocation' , 'https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation'