Final Report of Recipe Search Engine

Jinge Ma
University of Michigan
jingema@umich.edu

Yaning Zhang
University of Michigan
yaningzh@umich.edu

Zhaoying Pan
University of Michigan
panzy@umich.edu

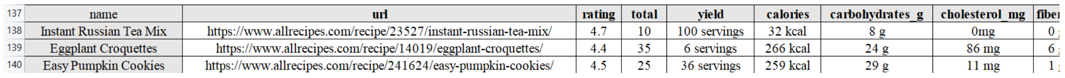
1 Introduction

Information retrieval (IR) can help users find information stored in a system, it is a huge challenge these days with booming information. We decided to make a recipe search system for food lovers like us. The objective of this project was originally to create a food search system based on user queries. However, upon further clarification of the definition of a search system, the goal was modified to the development of a recipe information retrieval system. This system accepts user queries including recipe name, main ingredients, rating, calories, and unwanted ingredients, and returns a list of relevant recipes ranked in order of relevance to the query. For most of the recipe search website, it can only search only based on the name of the recipe without considering other information like ingredients wanted and unwanted, nutrition info, etc., but ours can achieve that. Our system can let the user specify their requirements and we will find the most effective one to them. We make use of several weighting methods, BM25, TF-IDF, naive system and our own model, and several evaluation metrics, MAP, nDCG, and Precision. Our results have a high value on nDCG and MAP which indicate that it can meet user requirements as much as possible.

2 Data

2.1 Data Collection

All of our recipes are scraped from `https://www.allrecipes.com/recipes/`. In order to scrape this website, first we obtained the urls for all the recipes, then we can get the information from the web page for every singe recipe. All the recipes are stored in a single .csv file. Every row is corresponding to a recipe, including the name, url, ingredients, and nutrition, etc. Here is a screenshot of the dataset.

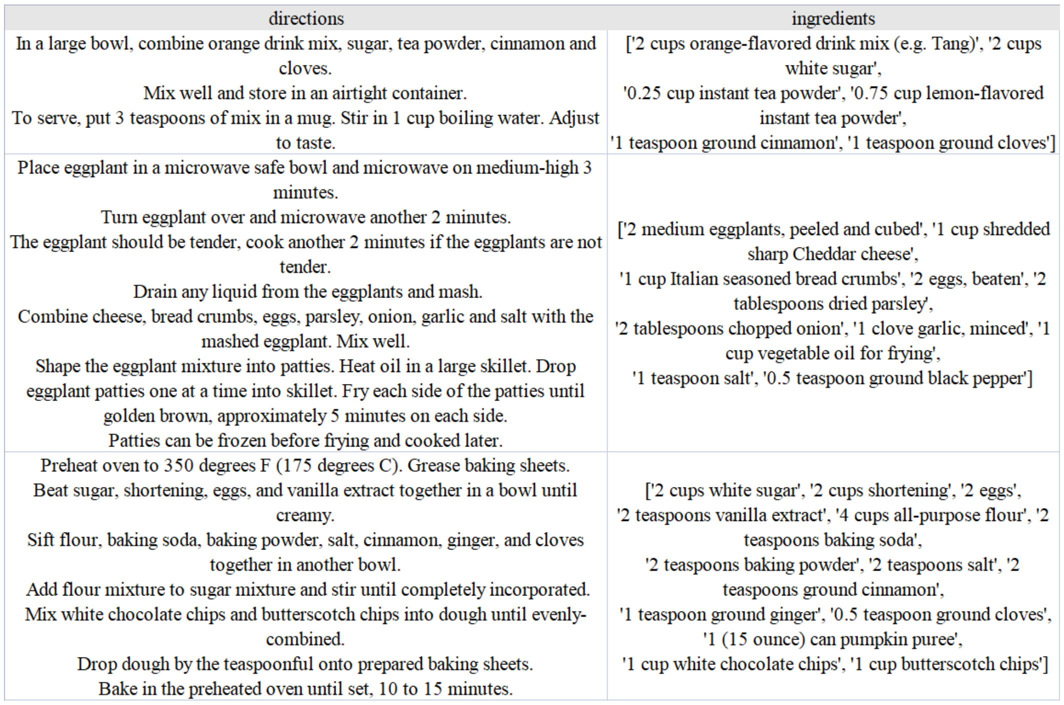


137	name	url	rating	total	yield	calories	carbohydrates_g	cholesterol_mg	fibe
138	Instant Russian Tea Mix	https://www.allrecipes.com/recipe/23527/instant-russian-tea-mix/	4.7	10	100 servings	32 kcal	8 g	0mg	0
139	Eggplant Croquettes	https://www.allrecipes.com/recipe/14019/eggplant-croquettes/	4.4	35	6 servings	266 kcal	24 g	86 mg	6
140	Easy Pumpkin Cookies	https://www.allrecipes.com/recipe/241624/easy-pumpkin-cookies/	4.5	25	36 servings	259 kcal	29 g	11 mg	1

Figure 1: A screenshot of the dataset: name, url, rating, total time, yield ...

The details is described as follows:
1. Using scrapy to parse and store structured data for all categories of recipes and corresponding urls from the start urls in list respectively. Start urls examples: `https://www.allrecipes.com/recipes/76/appetizers-and-snacks/?page=2`, `https://www.allrecipes.com/recipes/88/bbq-grilling/?page=2`,...
2. Loop through each start urls' list, keep using scrapy to parse every recipe's name and corresponding URL (such as `https://www.allrecipes.com/recipe/267865/spooky-spider-halloween-hot-dogs`), which should be stored in list.
3. Iterate through all recipes in the urls list to get all single recipe's basic information, such as the recipe's rating, ingredients, nutritions, etc.

Finally, we put all the information of recipes into a single 46469.csv file.



directions	ingredients
In a large bowl, combine orange drink mix, sugar, tea powder, cinnamon and cloves. Mix well and store in an airtight container. To serve, put 3 teaspoons of mix in a mug. Stir in 1 cup boiling water. Adjust to taste.	['2 cups orange-flavored drink mix (e.g. Tang)', '2 cups white sugar', '0.25 cup instant tea powder', '0.75 cup lemon-flavored instant tea powder', '1 teaspoon ground cinnamon', '1 teaspoon ground cloves']
Place eggplant in a microwave safe bowl and microwave on medium-high 3 minutes. Turn eggplant over and microwave another 2 minutes. The eggplant should be tender, cook another 2 minutes if the eggplants are not tender. Drain any liquid from the eggplants and mash. Combine cheese, bread crumbs, eggs, parsley, onion, garlic and salt with the mashed eggplant. Mix well. Shape the eggplant mixture into patties. Heat oil in a large skillet. Drop eggplant patties one at a time into skillet. Fry each side of the patties until golden brown, approximately 5 minutes on each side. Patties can be frozen before frying and cooked later.	['2 medium eggplants, peeled and cubed', '1 cup shredded sharp Cheddar cheese', '1 cup Italian seasoned bread crumbs', '2 eggs, beaten', '2 tablespoons dried parsley', '2 tablespoons chopped onion', '1 clove garlic, minced', '1 cup vegetable oil for frying', '1 teaspoon salt', '0.5 teaspoon ground black pepper']
Preheat oven to 350 degrees F (175 degrees C). Grease baking sheets. Beat sugar, shortening, eggs, and vanilla extract together in a bowl until creamy. Sift flour, baking soda, baking powder, salt, cinnamon, ginger, and cloves together in another bowl. Add flour mixture to sugar mixture and stir until completely incorporated. Mix white chocolate chips and butterscotch chips into dough until evenly-combined. Drop dough by the teaspoonful onto prepared baking sheets. Bake in the preheated oven until set, 10 to 15 minutes.	['2 cups white sugar', '2 cups shortening', '2 eggs', '2 teaspoons vanilla extract', '4 cups all-purpose flour', '2 teaspoons baking soda', '2 teaspoons baking powder', '2 teaspoons salt', '2 teaspoons ground cinnamon', '1 teaspoon ground ginger', '0.5 teaspoon ground cloves', '1 (15 ounce) can pumpkin puree', '1 cup white chocolate chips', '1 cup butterscotch chips']

Figure 2: A screenshot of the dataset: directions and ingredients

We get 46469 pieces of data for our project. But these data have some problems, because of the special characters in the name of the recipe, some of them cannot be encoded by UTF-8, so we need to clean it so that we can read the csv and start indexing.

Below is the step to clean our dataset:
1. Copy all the urls of the website, cause the urls have style of .../num/name, and the names are connected with $-$, so we can get the names of the recipe according to urls.
2. Read the csv file for url line by line and split the url by / character.
3. Pick the last element for the split list which is the name of the recipe connected by $-$, split the name by $-$ character and reconnect them by space.

2.2 Data Annotation

We have 60 queries (30 train, 30 test) and 4 kinds of methods in total, for each kind of method, we pick the top 50 results, and throw away all the duplicate data pieces. Up to now, we are only searching names of the queries, so we just compare the recipe names get from all methods with the query and manually give score of the results based on our cognition on a scale of 0-4. Then we convert the score to integer and delete all wasted columns except for docno, qid and label columns so that `pandas` datafram can successfully read our label result.

3 Related Work

For the recipe search engine that we are currently working on, there are some approaches that also give the recipe result based on their search engine, below are fives papers that are proceeded by others.

① *A Hybrid Semantic Item Model for Recipe Search by Example*[2]

This paper is written by Haoran Xie, et al. They thought that it is inadequate for traditional models to represent the characteristics of a recipe, they argued that a recipe model should be semantic-based and behavior-oriented, preferably with domain knowledge support. In this paper, a hybrid semantic item (HSI) model is developed based on cooking, eating, nutrition and media features to navigate recipes for people. In this paper, they are comparing between features but our model is searching by specific queries given by people in name, ingredients or nutrition fields. Their system is more likely a recommendation system while ours is more specific.

② *Overview of the NTCIR-11 Cooking Recipe Search Task*[5]

This paper is written by Michiko Yasukawa, et al. In this paper, they explore the information access tasks associated with cooking recipes with two types of users, professional and non-professional. This paper includes two subtasks, the ad hoc recipe search which means the user search for a recipe by natural language question, and recipe pairing which means users search for a complementary recipe for some query dish. Different from their paper, our paper will search through more fields not only name, or ingredients but also time, or nutrition information.

③ *Cooking Recipe Search by Pairs of Ingredient and Action — Word Sequence v.s. Flow-graph Representation*[3]

This paper is written by Yoko Yamakata, et al. They argued that the traditional searching method by using bags of words always results in misdetects. In this paper, by using the dependency parsing technique the text was converted to a flow-graph which can let the action sequence be easily extracted by tracing the path from the node corresponding to the ingredient to the root node corresponding to the last action. This method is like an improvement to the traditional methods, which can result in fewer errors, in the future we can have a try on this.

④ *Recipe search for blog-type recipe articles based on a user's situation*[1]

This paper is written by Takuya Kadowaki, et al. They established a system that finds recipes corresponding to the user's vague requirements. The system first extracts the author's reasons for the creation or selection of a recipe from the blog-type recipes, then it lets the user input their present situation or feelings which must include vague requirements and mine reasons for recipe selection from the input. Finally, the system outputs recipes that meet the user's vague requirements by associating the reasons for recipe selection from the user's input text with the reasons accompanying the blog-type recipes. Compared with the above method, our method will work on more data and with more specific categories cause we get data from a website instead of blogs.

⑤ *Concurrence of Word Concepts in Cooking Recipe Search*[4]

This paper is written by Michiko Yasukawa, et al. It is based on the regular words appear on the recipe. They parse the consistency of vocabulary and how agreement differs when searching through ingredients and dishes for common and uncommon items and between recipe authors and searchers. Compared with the above method, we do not care about the regular words, and also we do not compare the consistency between vocabularies. This is a field we can improve in the future.

4 Methodology

For the update of the project, we used four kinds of methods in our project which including BM25, TF-IDF, our own weighting function and also the naive weighting method generate by document length.

A. BM25 (Baseline)

In the simple system part, we implemented the untuned BM25 function as a simple-yet-effective method and our baseline.

We use `pyterrier Batchretrieve` method to implement the BM25 model. The scoring function for it is:

$$S(Q,D) = \sum_{t \in Q \cap D} \ln \frac{N - df(t) + 0.5}{df(t) + 0.5} \cdot \frac{k_1 + 1 \cdot c(t,D)}{k_1(1 - b + b\frac{|D|}{av_{dl}}) + c(t,D)} \cdot \frac{(k_3+1)\cdot c(t,Q)}{k_3 + c(t,Q)} \quad (1)$$

$c(t,D)$: term frequency in document
N: total number of documents in the collection
$df(t)$: number of documents contains term
$|D|$: length of the document
av_{dl}: average length of all documents

$c(t, Q)$: term frequency in query
k_1, k_3, b: hyperparameters for $k_1 = 1.2$, $k_3 = 8$, $b = 0.75$

B. TF-IDF

We also those TF-IDF as a system to get the score. We use `pyterrier Batchretrieve` method to implement the BM25 model. The scoring function for it is:

$$S = c(t, D) \cdot (1 + \log\left(\frac{N}{df(t)}\right)) \tag{2}$$

$c(t, D)$: term frequency in document
N: total number of documents in the collection
$df(t)$: number of documents contains term

C. Naive System

For the naive system, it works as the lowest possible bar for the performance, it is more likely a random process, and we choose document length for the result, which means $S = |D|$.

D. Our own weighting function

In the section, we also design our own weighting function, the function is designed based on the 5 constraints including term frequency, length normalization and TF-length mentioned in the lecture:

Constraint 1: If $q \in Q$ and $t \notin Q$, then $S(Q, D \cup \{q\}) > S(Q, D \cup \{t\})$
Constraint 2: Q be a query with only one term q, D_1, D_2, D_3 of same length, if $c(q, D_2) - c(q, D_1) = c(q, D_3) - c(q, D_2) = 1$, then $S(q, D_2) - S(q, D_1) = S(q, D_3) - S(q, D_2)$
Constraint 3: If $t \in T$ is a non-query term, then $S(D \cup \{q\}, Q) < S(D, Q)$
Constraint 4: If $D \cap Q \neq \emptyset$, and D_k is constructed by concatenating D with itself k times, then $S(D_k, Q) \geq S(D, Q)$
Constraint 5: If $q \in Q$, then $S(D \cup \{q\}, Q) > S(D, Q)$

Based on not violating these constraints, we designed our own function:

$$S(Q, D) = \sum_{t \in Q \cap D} c(t, Q) \cdot \frac{k \cdot c(t, D)}{k \cdot c(t, D) + ln\left(1 + \frac{|D|}{av_{dl}}\right)} \cdot \left(\frac{N+1}{df(t)}\right)^b \tag{3}$$

$c(t, D)$: term frequency in document
N: total number of documents in the collection
$df(t)$: number of documents contains term
$|D|$: length of the document
av_{dl}: average length of all documents
$c(t, Q)$: term frequency in query
k, b: hyperparameters for $k = 0.85$, $b = 0.45$

The term frequency in query is with no modification, cause it is normally 1, and does not influence much on the result. For the IDF part, we need to consider the partition of the document frequency, it also needs to be adjusted, I use power to reduce the influence. The value of b is usually between 0-1 and we choose 0.45 to reduce more on the influence. The length of document also needs to adjust, and it comes together with the term frequency in the document. The document length is given a $ln()$ outside it just to reward short documents and penalize long ones. For the term frequency in the document, it should work with the document length term just like BM25 with one numerator and one denominator together with document length part. A parameter is given to it and set to 0.85 cause we need to reduce little influence on it.

E. Adding features to our own function

As is mentioned above we have our own weighting function, but the performance is not beating the `BM25` baseline in our update, so we add some features to our own function to improve the performance of it. The weighting function is act as a initial feature passing into our features.

The first change is in indexing the text, instead of passing the `"docno"` as the only `meta_fields` we are adding the whole dataframe as the `meta_fields`. In our features we have retrieved a few columns of data to pass into our features, we use:

```
pt.text.get_text(recipe_index,["title","docno","rating","ingredients","
                                    instructions"])
```

We are adding 7 features in our whole features. The first one is the identity transformer, this can be useful for adding the candidate ranking score as a feature in for learning-to-rank. We have also add three frequently used weighting function to our features to improve the performance, they are `BM25`, `TF-IDF` and `CoordinateMatch` weighting functions, the reason why we do that is the results of these weighting functions are different if we can add some of them, our results can make a combination of them and it can be improved. The fifth feature we add is we made a constrain of ratings, we give score to the one greater than or equal to 4.5 for higher score means the recipe is more appreciate by users. We are also adding two features according to the `instructions` and the `ingredients` columns, we use the `BM25` model to calculate the score and pass it to the function, cause we think that the ingredients and instructions can in some ways indicate the name of the recipe, cause most recipes are named according to the ingredients they have and the instruction will also mention the ingredients. This act as a double protection to our model.

The final feature is like:

```
ltr_feats1 = (my_weighting) >>
    pt.text.get_text(recipe_index2,["title","docno","rating","ingredients
                                    ","instructions"]) >> (
    pt.transformer.IdentityTransformer()
    **
    bm25
    **
    tfidf
    **
    pt.BatchRetrieve(recipe_index2, wmodel="CoordinateMatch")
    **
    (pt.apply.doc_score(lambda row: int(row["rating"] is not None and
                                    float(row["rating"]) >= 4.5)))
    **
    (pt.text.scorer(body_attr="ingredients", takes='docs', wmodel='BM25')
                                    )
    **
    (pt.text.scorer(body_attr="instructions", takes='docs', wmodel='BM25'
                                    ))
)
```

For the above feature we have, we use three Learning To Rank (LTR) models which are `coordinate_ascent` model from `fastrank` package, `RandomForestRegressor` model from `sklearn` package and `LambdaMART` model from `lightgbm` package to do the training.

F. Composition and negation

In order to improve the effectiveness of the search engine, we implemented two functions: the composition function and the negation function. The composition function allows users to input multiple types of queries, such as cuisine name, desired rating range, calorie range, and desired ingredients, to receive more targeted search results. The negation function allows users to specify ingredients that they do not want to be included in the search results. To rank the cuisines in the dataset, the search engine calculates a relevance score for each cuisine based on each individual query, such as the recipe name, desired ingredients, and unwanted ingredients. For composition queries, the corresponding scores are added to the overall score, while for negation queries, the corresponding scores are subtracted. This approach allows for more nuanced searching and avoids the limitations of a traditional search method such as BM25, which cannot understand the concept of negation and may return results that include unwanted ingredients when they are included in the query.

Some possible strengths of the composition and negation functions are: 1. Increased specificity: By allowing users to input multiple types of queries, the composition function allows for more targeted search results. 2. Improved accuracy: The negation function allows users to specify ingredients that they do not want to be included in the search results, which can improve the accuracy of the search results.3. Flexibility: The composition and negation settings could be applied to other information retrieval algorithms in addition to BM25.

Some possible weaknesses of these functions are: 1. Complexity: These functions may require more complex algorithms and data processing, which can increase the complexity and searching time of the search engine. 2. Limited data: The accuracy of the search results may be limited by the quality and breadth of the data in the dataset.

Here is the pseudo code for the composition and negation function.

```
function CompositionandNegationSearch(queries):
  overallScore = 0
  for query in positive_queries:
    score = calculateScore(query)
    overallScore += score
  for query in negative_queries:
    score = calculateScore(query)
    overallScore -= score
  return overallScore
```

Here is an example that combines both the composition and negation functions:

Name: "Thai Curry"

Rating: 4-5

Wanted ingredients: coconut milk, curry paste, vegetables

Unwanted ingredients: fish sauce, shrimp

Using these queries, the search engine could return a list of Thai curry recipes with a rating of 4 or more stars that include coconut milk, curry paste, and vegetables as ingredients, but do not include fish sauce or shrimp. The composition function would take into account the desired ingredients and rating, while the negation function would exclude recipes that include the unwanted ingredients.

5 Evaluation and Results

As we discussed in the data section, we annotate 60 queries with retrieved results from four methods and split them into 20 queries for training 20 queries for validation, and 20 queries for testing. We use the `pandas` data frames of 20 testing queries and corresponding annotations of relevance as the topics and qrels for the `pyterrier` built-in experiments to compare our current implementation with two baselines and a naive system.

We apply BM25 and TF-IDF from `pyterrier` as our baseline, and we also develop a naive system returning retrieved results ranked by length to set the lowest possible bar for performance for comparison. Regarding the metrics, we use built-in Mean Average Precision (MAP), Precision at 5, normalized Discounted Cumulative Gain (nDCG), and nDCG at the rank 10 cutoff. We have the data annotated from 0-4 scores, and for the documents we do not annotate, the `pandas` will automatically add zeros to it. The formula for nDCG is:

$$nDCG = \frac{DCG}{IDCG} = \frac{rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i}}{rel_1^{ideal} + \sum_{i=2}^{p} \frac{rel_i^{ideal}}{\log_2 i}} \tag{4}$$

The formula for MAP is:

$$mAP = \frac{1}{N} \sum_{N}^{i=1} AP_i \tag{5}$$

We first perform the experiment on all 20 testing queries, and the results are shown in Table 1. The `Coordinate Ascent` model silghtly beat the baseline of `BM25` while others achieved comparable results compared to baselines and performed much better than the naive system.

model	MAP	P_5	nDCG	nDCG_cut_10
Coordinate Ascent	0.8887	0.98	0.9606	0.9487
Random Forest	0.7455	0.95	0.9021	0.8578
LambdaMART	0.7982	0.94	0.9225	0.8925
BM25	0.8870	0.98	0.9602	0.9423
TF-IDF	0.8870	0.98	0.9606	0.9423
Random System	0.1577	0.49	0.5225	0.3880

Table 1: Comparison of Our Method with Naive System and Baselines (BM25, TF-IDF)

Except for the overall experiment, we also evaluated the performance for each individual feature we used to train our model. The independent performance for each feature is shown in the figure below3:

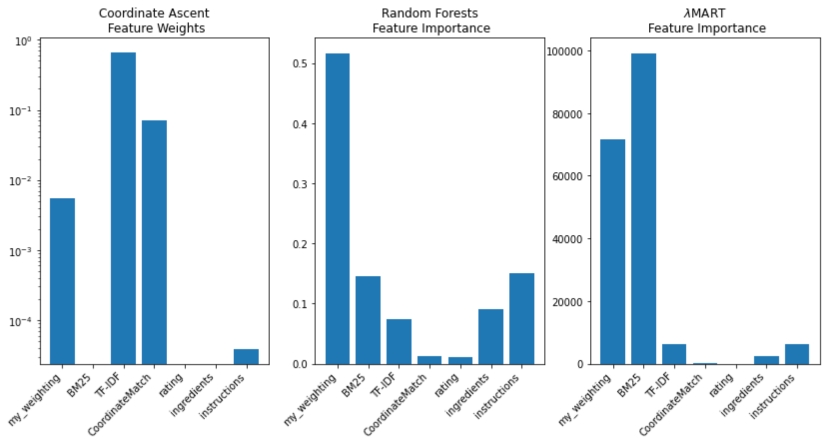


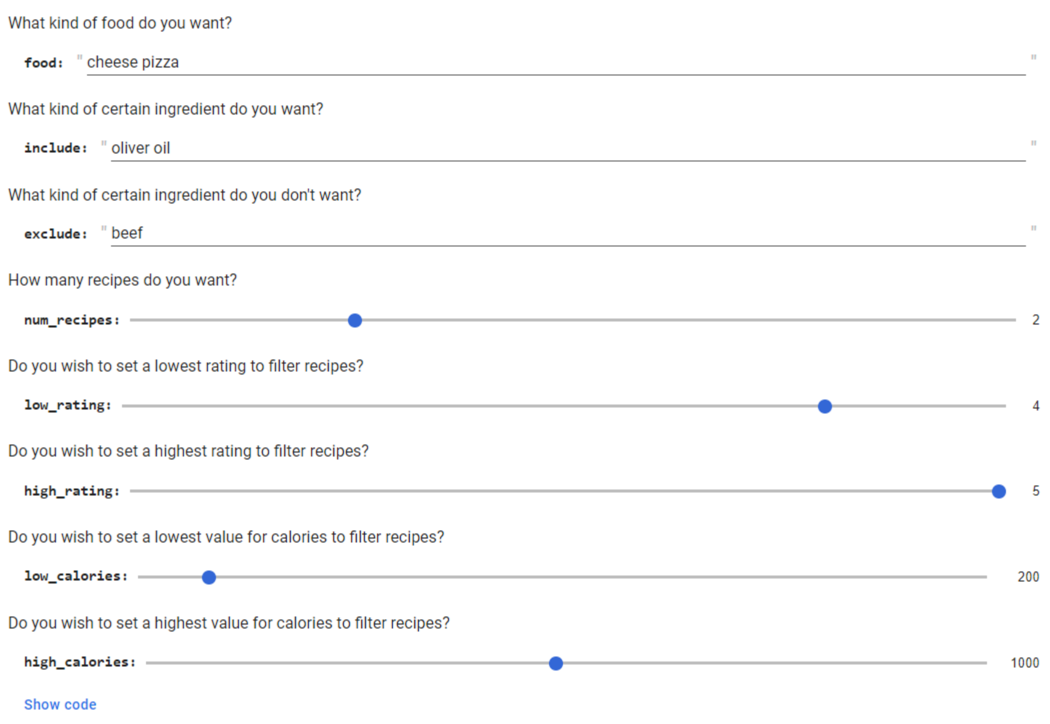
Figure 3: Feature Importance in the three models

Figures 4, 5, and 6 illustrate the interface of the search engine, the text descriptions for the top two most relevant recipes, and the images of these recipes, respectively. These figures demonstrate the ability of the search engine to return relevant and accurate results based on user queries.

6 Discussion

As you can see from the above part, our metrics all work well, we can let users search recipes according to names, ingredients, rating, nutrition information etc. separately, it can also achieve the request for not including items. This is very user-friendly, cause it can satisfied the users' requirements in different fields.

In our project update, the weighting function we have did not beat the baseline of `BM25`, after adding features to our own function and using the LTR models, one of our method, `coordinate_ascent` can slightly beat the baseline of `BM25` where MAP nDCG and nDCG@10 of `BM25` is 0.8870, 0.9602 and 0.9423 and our own function has a result of 0.8887, 0.9606 and 0.9487, and the others have comparable results. The reason of causing this is that the three LTR methods have different importance in features. Comparing these three, all of them have less independence on the "rating" column, which means this do not influence more on the results. For the `coordinate_ascent` model, it depends on `TF-IDF`, `CoordinateMatch` and our own weighting function and with less dependence on the `BM25` score of the "instructions" column. And nearly no dependent on other features, `TF-IDF` have a good performance as is mentioned in the result part, and this model depends a lot on it, that is why it has a higher score. For `Random Forest` it has the worst performance among all, it based a lot on our own weighting function which did not beat baseline, it should pay more attentions on other features which would generate a better result. For the `LambdaMART` model,


What kind of food do you want?

food: cheese pizza

What kind of certain ingredient do you want?

include: oliver oil

What kind of certain ingredient do you don't want?

exclude: beef

How many recipes do you want?

num_recipes: 2

Do you wish to set a lowest rating to filter recipes?

low_rating: 4

Do you wish to set a highest rating to filter recipes?

high_rating: 5

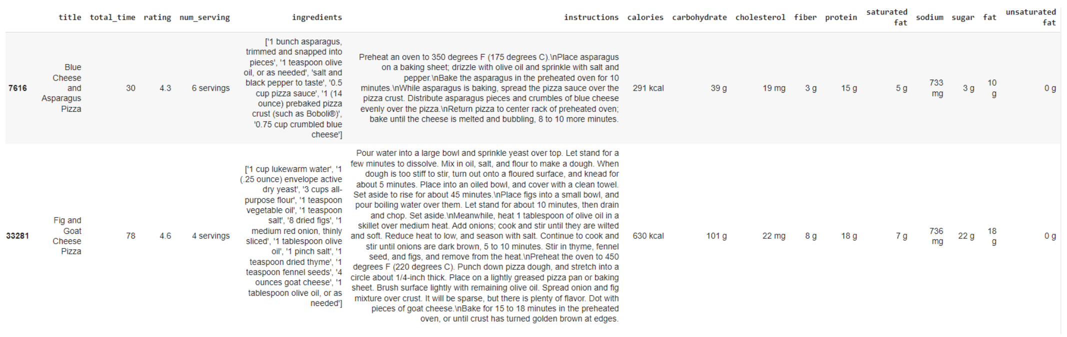
Do you wish to set a lowest value for calories to filter recipes?

low_calories: 200

Do you wish to set a highest value for calories to filter recipes?

high_calories: 1000

Figure 4: The user interface of our search engine with composition and negation functions



	title	total_time	rating	num_serving	ingredients	instructions	calories	carbohydrate	cholesterol	fiber	protein	saturated fat	sodium	sugar	fat	unsaturated fat
7616	Blue Cheese and Asparagus Pizza	30	4.3	6 servings	['1 bunch asparagus, trimmed and snapped into pieces', '1 teaspoon olive oil, or as needed', 'salt and black pepper to taste', '0.5 cup pizza sauce', '1 (14 ounce) prebaked pizza crust (such as Boboli®)', '0.75 cup crumbled blue cheese']	Preheat an oven to 350 degrees F (175 degrees C).\nPlace asparagus on a baking sheet; drizzle with olive oil and sprinkle with salt and pepper.\nBake the asparagus in the preheated oven for 10 minutes.\nWhile asparagus is baking, spread the pizza sauce over the pizza crust. Distribute asparagus pieces and crumbles of blue cheese evenly over the pizza.\nReturn pizza to center rack of preheated oven; bake until the cheese is melted and bubbling, 8 to 10 more minutes.	291 kcal	39 g	19 mg	3 g	15 g	5 g	733 mg	3 g	10 g	0 g
33281	Fig and Goat Cheese Pizza	78	4.6	4 servings	['1 cup lukewarm water', '1 (.25 ounce) envelope active dry yeast', '3 cups all-purpose flour', '1 teaspoon vegetable oil', '1 teaspoon salt', '8 dried figs', '1 medium red onion, thinly sliced', '1 tablespoon olive oil', '1 pinch salt', '1 teaspoon dried thyme', '1 teaspoon fennel seeds', '4 ounces goat cheese', '1 tablespoon olive oil, or as needed']	Pour water into a large bowl and sprinkle yeast over top. Let stand for a few minutes to dissolve. Mix in oil, salt, and flour to make a dough. When dough is too stiff to stir, turn out onto a floured surface, and knead for about 5 minutes. Place into an oiled bowl, and cover with a clean towel. Set aside to rise for about 45 minutes.\nPlace figs into a small bowl, and pour boiling water over them. Let stand for about 10 minutes, then drain and chop. Set aside.\nMeanwhile, heat 1 tablespoon of olive oil in a skillet over medium heat. Add onions; cook and stir until they are wilted and soft. Reduce heat to low, and season with salt. Continue to cook and stir until onions are dark brown, 5 to 10 minutes. Stir in thyme, fennel seed, and figs, and remove from the heat.\nPreheat the oven to 450 degrees F (220 degrees C). Punch down pizza dough, and stretch into a circle about 1/4-inch thick. Place on a lightly greased pizza pan or baking sheet. Brush surface lightly with remaining olive oil. Spread onion and fig mixture over crust. It will be sparse, but there is plenty of flavor. Dot with pieces of goat cheese.\nBake for 15 to 18 minutes in the preheated oven, or until crust has turned golden brown at edges.	630 kcal	101 g	22 mg	8 g	18 g	7 g	736 mg	22 g	18 g	0 g

Figure 5: Text descriptions for the top two most relevant recipes

it based a lot on `BM25` and our own weighting function, while lower dependence on `TF-IDF`, cause our own function did not better than `BM25`, it just like pull down the original performance.

We also suppose that if the feature importance of score of "ingredients" column would increase, the result would be better, cause most recipe names are raised based on the ingredients. And our model uses a lot of "BM25" scoring, it depends a lot on it, so it is hard to beat the baseline. In the future we may try some other weighting methods for the two columns so that it may performs better.

7 Conclusion

In this report, we have build a weighting function by ourselves according to the constraints mentioned above, we use it as a feature and pass it into our own models, we are adding several more features, like `BM25`, `TF-IDF`, `CoordinateMatch`, and the score of ingredients, and instructions ranking by `BM25`. We pass these features into three LTR models, `coordinate_ascent`, `RandomForestRegressor` and `LambdaMART` to do the training and get a comprehensive result. We also analysis the importance of each feature to the final result.

Figure 6: Images for the top two most relevant recipes

8 Other Things We Tried

8.1 Re-ranker

This part is achieved by using the package `onir_pt`, it is a open source from Georgetown Information Retrieval Lab, we have formed a `knrm` re-ranker:

```
knrm = onir_pt.reranker('knrm', 'wordvec_hash', text_field="title")
```

The pipeline is to use the `BM25` pass into the text field `"title"`, `"ingredients"` and `"instructions"` and then pass into the `knrm` re-ranker. We only try for the `BM25` pipeline considering about the time consuming and the device we have. After the pipeline is formed we use it to fit with the train and validation topics. The model fit for 63 iterations. For the final result we are using several methods, MAP, nDCG, nDCG@10 and P@10 to compare it with the `BM25` baseline, the final result is in table2.

model	MAP	nDCG	nDCG@10	p_10	mrt
BM25	0.9031	0.9638	0.9419	0.9767	190.67
knrm	0.7348	0.8718	0.7867	0.8633	249.97

Table 2: Comparison of knrm re-ranker with BM25 Baselines

As we can see from the above table2, the result of the re-ranker has lower value than the result of our `BM25` baseline. So we decided not to use this. I am not quite sure about why this happens, cause the results in the homework also not beating the baseline. I suppose that this should beat the baseline, cause it is using both the training data and the validation data to validate it, which may have more precision.

9 What's Next

Due to the time limitation, we did not do anything on query, as is mentioned in the lecture, query expansion is a technique used to improve the effectiveness of a search by adding additional terms to the original query, we can calculate the pairs or threes frequency of words appears in the name of the recipe and adding some words to the original query according to the frequency we have. We can also use is the query log analysis, the main idea is log the user behaviours or actions in we search, for which we can build personalize system for users.

Due to the lime limitation and the device we have. We did not try anything related to deep learning. It is also a effective method to improve the search system we have, we can tune parameters or adding features by using this. In the future, we can add relevant score in the our feature by training plenty of data with the query we have and add the relevance score as a feature to our model. We need to collect and annotate more data so that the result we train is accurate enough.

References

[1] Takuya Kadowaki et al. "Recipe Search for Blog-Type Recipe Articles Based on a User's Situation". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. UbiComp '14 Adjunct. Seattle, Washington: Association for Computing Machinery, 2014, pp. 497–506. ISBN: 9781450330473. DOI: 10.1145/2638728.2641327. URL: https://doi.org/10.1145/2638728.2641327.

[2] Haoran Xie, Lijuan Yu, and Qing Li. "A Hybrid Semantic Item Model for Recipe Search by Example". In: *2010 IEEE International Symposium on Multimedia*. 2010, pp. 254–259. DOI: 10.1109/ISM.2010.44.

[3] Yoko Yamakata et al. "Cooking recipe search by pairs of ingredient and action—word sequence vs flow-graph representation—". In: *Transactions of the Japanese Society for Artificial Intelligence* 32.1 (2017), WII–F_1.

[4] Michiko Yasukawa and Falk Scholer. "Concurrence of Word Concepts in Cooking Recipe Search". In: *Proceedings of the 9th Workshop on Multimedia for Cooking and Eating Activities in Conjunction with The 2017 International Joint Conference on Artificial Intelligence*. CEA2017. Melbourne, Australia: Association for Computing Machinery, 2017, pp. 25–30. ISBN: 9781450352673. DOI: 10.1145/3106668.3106674. URL: https://doi.org/10.1145/3106668.3106674.

[5] Michiko Yasukawa et al. "Overview of the NTCIR-11 Cooking Recipe Search Task." In: *NTCIR*. 2014.