QuicklyCook: A User-Friendly Recipe Recommender

Naila Bushra and Mehedi Hasan

Mississippi State University, Mississippi State MS 39759, {nb921, mh3092}@msstate.edu

Abstract. We present 'QuickyCook', an instance-based recipe recommender system. It uses a recently developed database called 'Recipe 1M' which contains 1 million recipes for a variety of food items. The proposed system is developed by taking advantage of the fast execution of available python data types and libraries which greatly enhance the proposed model's performance. The system is built off the idea of a very popular algorithm i.e. K-nearest neighbor (KNN) where a new instance is classified based on the classes of already-seen instances. However, we have proposed a slightly different and unique way of calculating the similarity score of the new to-be-classified instance with the previous ones. The proposed system allows the user to input a unrestricted list of ingredients and finds the closest 'k' recipes containing preferably all of the input ingredients. Unlike many recipe recommender systems to date, our proposed system does not restrict the user to choose ingredients from an existing ingredient list thus providing more flexibility to the user. Performance-wise, its execution time is considerably faster as it does not require large feature vectors to be formed and compared for classification. Hence, QuicklyCook can be considered as a simple but powerful application.

Keywords: K-nearest neighbor · recipe recommender · sequence matching · text comparison · Recipe 1M

1 Introduction

Food datasets are an interesting area that can be explored to extract a variety of information. The images of food are commonly used to recognize meal type, ingredients list, predict and learn food habits and the variability of food habits across cultures and places. Also, food recipes are used to predict the name of the food, ingredient list is used to predict the recipe, learn about food patterns of different countries, and so on.

We propose a recipe recommender system in this paper called **QuicklyCook** where given a list of ingredients as inputs, the system will suggest recipes based on those ingredients. The target users of this system are the ones who have necessary ingredients in their hands to make an appropriate meal but lacks references or ideas on how to do so. The main motivation for the proposed idea is to benefit people by reducing their valuable time and money as well. A lot of time is wasted on the internet to read recipes and watch cooking shows on media. People also spend a lot of their valuable time reading unnecessary detailed cookbooks claiming to contain ‘all the recipes’ of the world. Moreover, in addition to wasting time people when hungry run to restaurants or fast food shops to grab a quick bite that may not always be quick considering the travel and waiting time. Most of the time eating outside can be a negative factor for people who are trying to watch their health. In most of the cases mentioned here, there is a great possibility that these people already have enough ingredients in their kitchens to make a meal in a surprisingly quick amount of time. These people will then be able to save a good amount of time and money to do other things that are of more priority to them. Thus the proposed system can be proven to be very much useful for people who do not want to waste time or money on food. QuicklyCook applies the simplest ideas to suggest recipes to users based on the readily available ingredients they have in their kitchens.

The rest of the paper is organized as follows; Section 2 contains the discussion of different research work related to food dataset especially recipe recommendation systems. Section 3 contains the proposed idea in a nutshell and the list of features that our system has. Section 4 contains the overview of the resources that are used in the proposed system including the Recipe 1M dataset [10] followed by section 5 that contains a detailed description of the methodology used to develop the system (data processing, model training, and prediction). Section 6 lists the results and performance analysis of the system. Section 7 describes the limitation of this paper and the possible future works to improve the current system.

2 Discussions

Before building our system we have explored existing web-based recipe recommender systems as well as the ones in the literature that provide similar service proposed by our system. A small list of several online websites that we have looked into is given in Appendix [A]

The goals of the existing recipe recommender system are diverse. Each recommender system fulfills the specific needs of the user of the system. A lot of them emphasize on providing the user with healthy recommendation [2] [4] [7] [11]. This type of recipe recommender ultimately helps with several health-related issues such as obesity. However, they do not provide the option for the user to selectively choose healthy ingredients of their preference.

Techniques such as folksonomy, similarity measures, ingredient tags, and latent factors computation, content-based, semantic understanding, etc. are widely uses to provide the user with the best experience with the recommender system [12] [8] [3]. Collaborative filtering is another popular technique used in a lot of popular recommender system that suggests products (in our case recipes) to users based on their similar preferences with other users [1] [2]. The goal of

QuicklyCook is more inclined towards giving the user more personalized, specific, and fewer number of recommendation as possible.

These existing websites mentioned above have a range of constraints that lie between a hungry user and his quickly cooked meal. The limitations of these services are given in the following subsections:

– **Restricted Ingredient List** Most of the time, the ingredients that the user tries to enter have to be chosen from a list provided by the website. Sometimes, the desired ingredient may not even be on that list. E.g. recipeland.com.
– **Lengthy Steps** Sometimes users have to search through a long list to find his/her desired ingredient. On most of the website, the user has to go through certain steps to define the type of ingredient, its amount and other unnecessary customization which delays the process by making it complex. E.g. ideas4recipes.com.
– **Too Many Results** The list of suggested recipes is usually too long for the user to search thoroughly to find the one that the user really wants to cook at that time. E.g. bigoven.com
– **Missing Essential Customization** In relation to the previous limitation, on several websites that we have explored, there were no customization options like preparation time or cuisine type or food type. These filtering options are pretty essential for the user to limit the results to the most intended ones. E.g. recipekey.com.
– **Recipes With Additional Ingredients** One of the reasons for their long result list is that they include recipes in the results which have a lot of additional ingredients that the user does not have and also did not give as inputs. If the user does not have the majority of the listed ingredients, he/she is most likely to avoid using that corresponding recipes. E.g. ideas4recipes.com
– **Limitation of The Number of Input Ingredients** The websites often limit the list of ingredients that is entered by the user from one to five which also limits the options for the user to try on more versatile recipes. E.g. ideas4recipes.com.

3 Proposed Idea

The proposed system is built off the idea used in the K-nearest neighbor (KNN) algorithm [6]. QuicklyCook is developed using python programming language and suggests recipes based on the ingredients that the user enters as inputs. This system uses the recently developed 'Recipe 1M' dataset [10] containing 1 million food items' recipe (categories shown in Figure 2). 70% of this dataset (700,000 recipes) is used to train our model and the other 30% (300,000 recipes) is used to test the model. This 70:30 split ratio allows the model to be trained better and learn about the data pattern before it starts to classify the never-seen-before data items in the test set. If we choose less ratio of data items for training, the model maybe under-trained and if we choose more than it may over-fit the data and completely underperform on the test dataset. he training model forms a feature vector for each of the recipes. The features here are the ingredients of the recipe. The unlabeled list of input ingredients are compared with the feature vectors of the training data to output related recipes. The recipes that contain all the ingredients entered by the user are returned by the system and suggested to the user. If there are multiple recipes containing all the ingredients entered by the user then preference is given to the recipes that have 'fewer' additional ingredients. Several python libraries such as 'pandas' [5] and 'difflab' [9] have been used for faster manipulation of the dataset. The major features of the system are described in the following subsections:

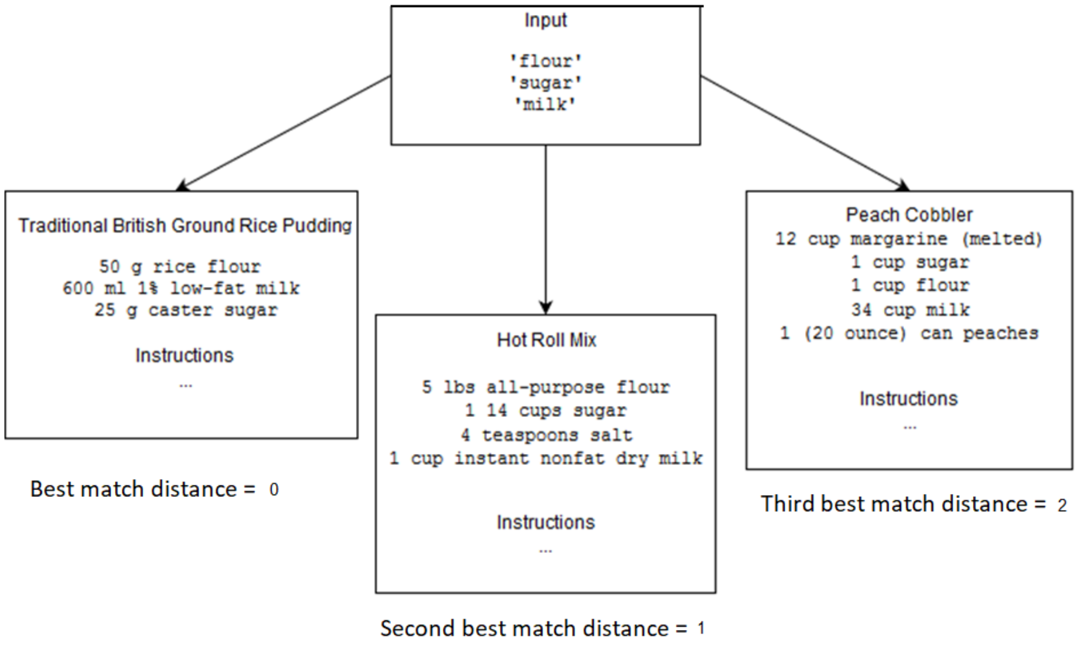

Fig. 1: Suggested recipes for K = 3

- **Unrestricted Ingredient List** The user can enter as many ingredients as he/she wants according to their preference. The number of ingredients is not restricted and also the user does not have to choose the ingredients from an existing list. The user can also ask for recipes that only contain a subset of the entered ingredients.
- **Auto-fix Typos** The system allows the user to make typos. In such cases, the system is still able to retrieve the recipe containing the corrected name

of the input ingredient by finding the closest correct match to the misspelled word.

- **Differentiate Ingredients** The system is able to understand the difference between tomato and tomato sauce, chili and chili powder and so on to find the appropriate recipe for the user.
- **Tune-able Similarity** It is possible to set the threshold that will determine how similar two ingredients are. While in the default setting the match needs to be an exact one (banana should not match with bananas), these constraints can be relaxed to suit the scenario.
- **'K'-recipes** Based on the user's preference, the system will result in 'K' nearest recipes that can be cooked with the input ingredient list. The result is increasingly ordered by the number of additional ingredients.
- **Food Sensitivity** A large number of people have sensitivity for a certain type of food. For example, lactose intolerant consumers cannot consume food originated from dairy. Several have allergies of food like nuts, shrimp, eggplant, etc. The process flow of QuicklyCook allows filtering of food items not preferred by the user.

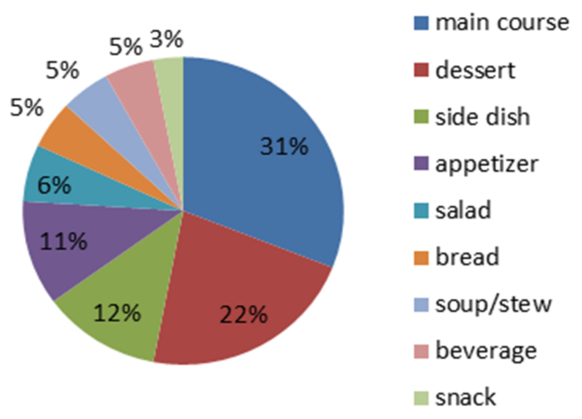

Fig. 2: Percentage of recipes based on categories in Recipe 1M

4 Data Sources

For the purpose of building our application, we have explored a range of available datasets of food recipes with ingredients and images of food as well. We have come across an outcome of pretty recent work by the researchers of the CSAIL lab of MIT where they have built up a database of various kinds of recipes from AllRecipes.com and Food.com websites. The database is called 'Recipe 1M' and contains 1 million recipes and 800,000 images of food [10].

The researchers who built this ‘Recipe 1M’ database have used Artificial Neural Network (ANN) to detect recipes from food images. For our research, we have used the recipe part of the dataset that is stored in a JSON file format (data stored as key-value pair). The size of this JSON file is around 1.3 GigaByte. To reduce the memory load we have divided this JSON file into eleven smaller JSON files. Each of them having the size around 130 MegaByte. Each of these smaller JSON files contains around 100,000 recipes. The key-value structure of the data is given in figure 3. This structure is maintained for each of the 1 million recipes.

```
{
        "ingredients":"…"
},
{
        "url":"…"
},
{
        "instructions":"…"
},
{
        "partition":"…"
},
{
        "title":"…"
}
```

Fig. 3: The format of recipe data in Recipe 1M

- Ingredients: The list of ingredients for the recipe (nested JSON)
- Url: Webpage address containing the detail of the recipe
- Instruction: Step by step instructions for cooking the item (nested JSON)
- Partition: indicates whether this particular recipe is a part of a training dataset, test dataset or validation dataset
- Title: Name of the food item

5 Methodology

5.1 Data Pre-processing

In the data pre-processing stage, we have divided the large JSON file containing the 1 million recipes into eleven smaller JSON files. Each of them contains around 100,000 recipes. To reduce the data size we have initially removed the ‘URL’s (not used in QuicklyCook) from all the eleven input JSON data files and converted them to ‘CSV’s. This reduces each of the file sizes to 20% (from 130 MB to 106

MB). The 'CSV' files have the following headers: id, ingredients, instructions, partition, and title.

The eleven 'CSV' files are then read into 'pandas' dataFrame (python datatype) [5] and combined to form the unified train data containing 70% of the dataset (700,000 recipes). The process of combining the eleven files was done into two stages to avoid the memory outrun issue. In the first stage, five and six of the eleven files were combines separately to form two intermediate files. The sizes of these two files were 583 MegaByte and 618 MegaByte. Finally, these two files were combined to get the final one containing 1 million recipes with a size of about 1.17 GigaByte. 70% of this 1 million recipes were then sent to the training model.

5.2 Formation of Feature Vector

5.3 K-Nearest Neighbor Algorithm

K-nearest neighbor is an instance-based learning algorithm. Here, the instances lie in an n-dimensional space. The training instances have their feature vectors of n-length and these instances are already classified. These instances are the recipes of our training data that already have their titles and details. Usually, for forming feature vector, a large set-based n-length feature vector is considered. In our case, that would be all possible n ingredients in the world (or at least in our whole database). The training instances will fill in 1 or 0 in that large feature vector depending on whether that particular ingredient is needed to make that recipe or not. When a new instance comes, it should form its own feature vector of length n. Then it is possible to compare these two vectors adopting a number of distance and similarity measures (i.e Hamming distance, Euclidean distance, Cosine Similarity etc.). This also requires natural language processing to some extent.

As our problem cannot be totally classified as a classification problem we have avoided forming huge feature vectors for finding similarity between input and training ingredients which is described in the next section.

5.4 Distance Measures

The ingredient list for 'Recipe 1M' dataset does not contain only single words. The list comes with the measurement as well (i.e. 1 cup of sugar). Therefore, we have split each of these ingredients into parts ('1', 'cup', 'of', 'sugar') to compare it with our input ingredient ('sugar'). The reason for doing that is the library function that we have used for the purpose of comparison (difflab.get_closer_match()) is not able to compare where there is empty space in a string is present. Also, it is unable to find the exact match between 'Sugar' and 'sugar' because of the cases. Therefore, we converted both the input and the training dataset to lower case before computing the distance using euclidean distance measure.

Afterward, the system gives us a list of possible match list (i,e [‘sugar’, ‘sugur’, ‘sogar’] from the training dataset compared to the input ingredient (‘sugar’). Now, we can decide whether to let the system give us an exact match or the closest match. This is measured by another library function (Sequence-Matcher.ratio()) and gives us 1.0 in case of an exact match. After that, all the recipes that have exact matches for all the input ingredients that were entered are returned. This returned list of recipes is ordered by the lowest number of additional ingredient that is needed by the user (Figure 1). Here, K determines how many closest match will be returned by the system,

6 Result and Performance

The execution time of several steps of the algorithm are given in Table I. The experiments were performed in an Intel Core i7 (quad-core) 2.6 GHz computer.

Table 1: Pre-processing and execution time of different process (Environment: Intel Core i7 .6 GHz, 8 GB Ram, editor: pycharm, python: 3.6.2).

Processes	**Time(in seconds)**
Partitioning the 1.3 GB json file into 11 parts	3
Converting 11 JSON files to CSV	7 (avg)
Combining CSV File 1 to 5 into one (583 MB) - Pass 1	27
Combining CSV File 6 to 11 into one (618 MB) - Pass 2	38
Combining CSV files from pass 1 and 2 to one final CSV	73
Training Model using 1Million dataset (1.3 GB)	129
Predicting using 1Million dataset (1.3 GB)	695
Training using combined file from pass 1 (583 MB)	61
Predicting using combined file from pass 1 (583 MB)	435
Training using smaller CSV file (106 MB)	11
Predicting using smaller CSV file (106 MB)	84

```
Input ingredients: ['flour', 'sugar', 'milk']

matched_recipe_indexes = [28, 49, …, 69765, 69784]

OrderedDict([(0, 51586), (1, 50345), (2, 69029),
(3, 69648), (4, 69140), (5, 69749), (6, 69724),
(7, 69744), (8, 69519), (9, 69723), (10, 69765),
(11, 69784), (12, 69382), (13, 69066), (14,
69103), (15, 68111), (16, 68593), (17, 69473),
(18, 65528), (19, 65905), (20, 68463), (21,
66403), (22, 61850), (23, 51307), (24, 62925),
(25, 60688), (26, 25554), (27, 68166), (29,
38543), (30, 49304), (35, 68927), (37, 38427),
(40, 43439)])

You can make ->
Traditional British Ground Rice Pudding

Ingredients:
->  50 g rice flour
->  600 ml 1% low-fat milk
->  25 g caster sugar

Instructions:
->  Preheat oven to 200 Centigrade / 400
Fahrenheit / Gas Mark 6 and grease a bake-proof
dish.
->  Heat milk in a saucepan and sprinkle on the
Ground Rice.
->  Stirring continuously, heat on a low heat
bring to a boil until it thickens.
->  Simmer for a further 2-3 minutes.
->  Stir in the sugar (My mum would have probably
used a bit more than 25g - sweet tooth!)
->  Pour into greased bake-proof dish.
->  Bake for 25-30 minutes until lightly brown.
```

Fig. 4: Sample output of QuicklyCook

We have mentioned previously the proposed system cannot be totally described as a classification system. Therefore, the accuracy depends on the implementation not on training the model. The performance, however, is a great concern as the dataset that is used is quite large in size. A sample output for the system **QuicklyCook** is given in figure 4. The processing time compared to the size of the database is quite impressive after the initial run. However, it still has some room for improvement.

7 Conclusion and Future Work

This proposed system has a lot of scope for improvement in terms of adding more data (from various countries' cuisine) to enhance user experience. It is a work in progress with the plan to extend this system to a web-based service where the user can actually try out and get benefit from the system. More control to the user can be provided by letting users add filters based on meal (appetizer/dessert/entre) type, cuisine type, preparation time, etc. It may also contain options that will let the user mark an ingredient as 'must-be-present' in the resulted recipes.

Another crucial improvement that can greatly increase performance is completely removing unnecessary words in the ingredient list which do not play any part during the similarity calculation (i.e. '1', 'cup', 'of', 'teaspoon', 'ounces', symbols, etc.)

Appendix A

- http://www.supercook.com
- http://www.myfridgefood.com
- http://www.foodpair.com
- http://www.bigoven.com
- http://www.recipepuppy.com
- https://recipeland.com
- http://www.recipekey.com
- http://www.ideas4recipes.com
- https://foodcombo.com

References

1. Berkovsky, S., Freyne, J.: Group-based recipe recommendations: analysis of data aggregation strategies. In: Proceedings of the fourth ACM conference on Recommender systems. pp. 111–118. ACM (2010)
2. Freyne, J., Berkovsky, S.: Intelligent food planning: personalized recipe recommendation. In: Proceedings of the 15th international conference on Intelligent user interfaces. pp. 321–324. ACM (2010)
3. Ge, M., Elahi, M., Fernaández-Tobías, I., Ricci, F., Massimo, D.: Using tags and latent factors in a food recommender system. In: Proceedings of the 5th International Conference on Digital Health 2015. pp. 105–112. ACM (2015)
4. Ge, M., Ricci, F., Massimo, D.: Health-aware food recommender system. In: Proceedings of the 9th ACM Conference on Recommender Systems. pp. 333–334. ACM (2015)
5. McKinney, W.: pandas: a foundational python library for data analysis and statistics. Python for High Performance and Scientific Computing pp. 1–9 (2011)
6. Michalski, R.S., Carbonell, J.G., Mitchell, T.M.: Machine learning: An artificial intelligence approach. Springer Science & Business Media (2013)

7. Mika, S.: Challenges for nutrition recommender systems. In: Proceedings of the 2nd Workshop on Context Aware Intel. Assistance, Berlin, Germany. pp. 25–33. Citeseer (2011)
8. van Pinxteren, Y., Geleijnse, G., Kamsteeg, P.: Deriving a recipe similarity measure for recommending healthful meals. In: Proceedings of the 16th international conference on Intelligent user interfaces. pp. 105–114. ACM (2011)
9. Rossum, G.: Python reference manual (1995)
10. Salvador, A., Hynes, N., Aytar, Y., Marin, J., Ofli, F., Weber, I., Torralba, A.: Learning cross-modal embeddings for cooking recipes and food images. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3068–3076. IEEE (2017)
11. Yang, L., Hsieh, C.K., Yang, H., Pollak, J.P., Dell, N., Belongie, S., Cole, C., Estrin, D.: Yum-me: a personalized nutrient-based meal recommender system. ACM Transactions on Information Systems (TOIS) **36**(1), 7 (2017)
12. Yu, L., Li, Q., Xie, H., Cai, Y.: Exploring folksonomy and cooking procedures to boost cooking recipe recommendation. In: Asia-Pacific Web Conference. pp. 119–130. Springer (2011)