COEN 281
GROUP 3 PROJECT REPORT
Ingredient Recommendation System for Recipes

Gaurav Karki (W1563134), Mega Putra (W1059479), Mayank Verma (W1538058)

SECTION I: INTRODUCTION

MOTIVATION

Eating is an essential part of our lives. For centuries, our ancestors have been developing recipes to not only eliminate their hunger, but also to create satisfying, delicious, and exciting meals. Cooking at home has been something that is gaining popularity since the early 2000's. The most significant increase since then was the percentage of men that cook increases from 37.9% to 51.9%. There has also been a 5% increase for women that cook at home, given historically, since women tend to cook more than men, this increase is noticeable. Given the overall increase of interest and how important food is to our lives, we wanted to create a project revolving around food, and cooking food.

We believe that the more people cook, the more likely they are to care about combining ingredients that go well together. In this day and age, we collect data, manipulate it, and use it to formulate decisions. We want to use the technology that we learned in hopes to provide ease in the kitchen for homecooks through data science, in this case, recommendations! Recommendation systems play a huge part in our day to day lives and it is sometimes taken for granted. Various services such as Netflix, Amazon, Spotify, IMDB, YouTube use recommendation systems to suggest what the user might like to 'consume' - creating a unique experience that is tailored for each user. Thus, greatly enhancing the user experience and promoting business for the company.

For our project, we want to build an easy-to-use recommendation system, which would suggest what ingredient(s) would go well with a set of user-selected ingredient(s). Our recommendation system will recommend multiple items from the set of ingredient(s) that the user enters, and based on those recommendations, the user will select more ingredients that he/she wants to add to their recipe. This allows homecooks to adjust their recipes based on the ingredients they have on the pantry given the multiple recommendations that our model provides. Moreover, this is more convenient to the users because they do not have to search for a specific recipe on the internet/cookbooks because there is a possibility that they might miss one or more ingredients from that recipe. Homecooks can simply input the ingredients they have in hand and our model will recommend multiple ingredients to choose from to create delicious meals.

OBJECTIVE

The objective of this project is to apply data science technologies to support our day to day lives. We gather, process, and mine data to find patterns that might be useful for the user. In this case, providing an easy-to-use ingredient recommendation system for homecooks, from patterns that our model generates. By using our application, homecooks can simply enter the ingredient(s) that they currently have in the pantry and our model will show multiple ingredients that go well with it. Then, the user can choose the next ingredient he/she will incorporate to the recipe. Since our model outputs multiple ingredients, users have an option of what to pick based on what they have on hand.

MARKET REVIEW

Allrecipes.com, which is one of the dataset that we use has around 1.5 billion visits annually, this shows that there is a high interest in home cooking and sharing recipes. Moreover, there are around 18 million cookbooks sold just in the United states in 2018, which is a 21% increase since 2017 (source: NPD Bookscan Survey 2018); and from our motivation above, we found that more and more people are cooking at home. Based on these facts, we think that an ingredient recommendation system could attract the market. We found several similar recommendation systems based around food and recipes, but none of them use the same algorithm or perform the same functions. Here are a few examples: there are models that recommend recipes based on ingredients and reviews, and there are other models that recommend recipes based on user-input ingredients, another one is an ingredient recommendation system that recommends similar ingredients, which is a substitution used when you don't have a certain ingredient. We believe that our model is beneficial, unique and can capture the market's attention.

SECTION 2: SYSTEM DESIGN AND IMPLEMENTATION DETAILS

ALGORITHM SELECTED

Since this is a recommendation system problem, we tried using the most straight forward algorithms such as KNN and collaborative filtering. However, we realized that our dataset consists of only recipes, which means that we do not have any knowledge about the user. Moreover, our goal is not to recommend ingredients based on user preference or popularity. We want to recommend ingredients that go well with the ingredient(s) that the user entered regardless of their preference. Therefore, in order to reach our goal we decided to go with Association Rule Mining using Apriori.

Association Rule Mining is a rule based machine learning algorithm that can uncover hidden structures within a transactional data, in this case recipes. It can determine if every ingredient has a number of items associated with it. In order to achieve that, we used the Apriori algorithm from the `efficient_apriori` library. This library contains an efficient and well-tested implementation of the Apriori algorithm as described in the original paper by Agrawal et al published in 1994. Figure 1 below shows an example of how the rule generation works.

Recipe	Ingredients
Chicken Curry	[Chicken Breast, Garam Masala, Salt, Pepper, Tomato, Onions, Chili, Garlic]
Margherita Pizza	[Mozzarella Cheese, Flour, Olive Oil, Dough, Tomato, Salt]
Beef Wellington	[Beef, Mushroom, Salt, Pepper, Onion, Egg, Pastry, Ham, Ground Meat, Garlic, Thyme, Butter]

Rule generated: {Salt, Pepper} -> {Onion}

Fig 1: Recipe Rule Generation Example

Apriori uses a breadth first search strategy to count the number of support itemsets and uses candidate generation function. Apriori utilizes a level-wise approach where it generates patterns containing 1 item, then 2 items, then 3 items, and so on. For our algorithm specifically it works on two phases. The first phase is to iterate over the transaction several times to build up the itemsets given the support level. The second phase is to build the association rule of the desired confidence, given the itemsets found in the first phase. “The Apriori prunes the search space efficiently by deciding a priori if an itemset possible has the desired support, before iterating over the entire dataset and checking.” - efficient apriori documentation.

TECHNOLOGY & TOOLS

In this project, we used Python3 along with integrated libraries such as `numpy`, `pickle`, `efficient_apriori`, and others. We decided to go with `numpy` because they are essentially like C/C++ arrays and have all the performance advantages. Also `numpy.load` feature is very powerful for reading a large dataset as it allows loading the numpy array into the memory and can be reused again. We used `pickle` to send huge nd arrays, objects, and dictionaries as binaries from the back end to the front end through our API.

To enable cooperative and agile workflow, we used `git` to collaborate the code from each one of our team members. The user interface was created using HTML, CSS, bootstrap3 and JavaScript. We used *flask* framework to create a Python web application since we are mainly using Python. The flask framework provides all the API endpoints, which a backend must implement, for example: GET and POST HTTP methods.

The database we are processing consists of more than 1 million recipes and around 3500 unique ingredients. Due to immense volume of data, it was essential to execute the algorithms on the high performance computer (HPC) that is available on campus. By using the HPC, we are able to take advantage of its larger on-demand memory and advanced CPU and GPU units.

SYSTEM DESIGN

Our project can be divided into two main parts: Core Data Mining and Python Web Application. The first part consists of a python script, which loads the data set, pre-processes the data, and executes the apriori algorithm. Due to the large quantity, our personal laptops were very slow in generating the results. We utilized the high performance computer (HPC) available at Santa Clara University, we ran the batch scripts, which ran overnight and stored the results in compressed files. Figure 2 below shows our system architecture.

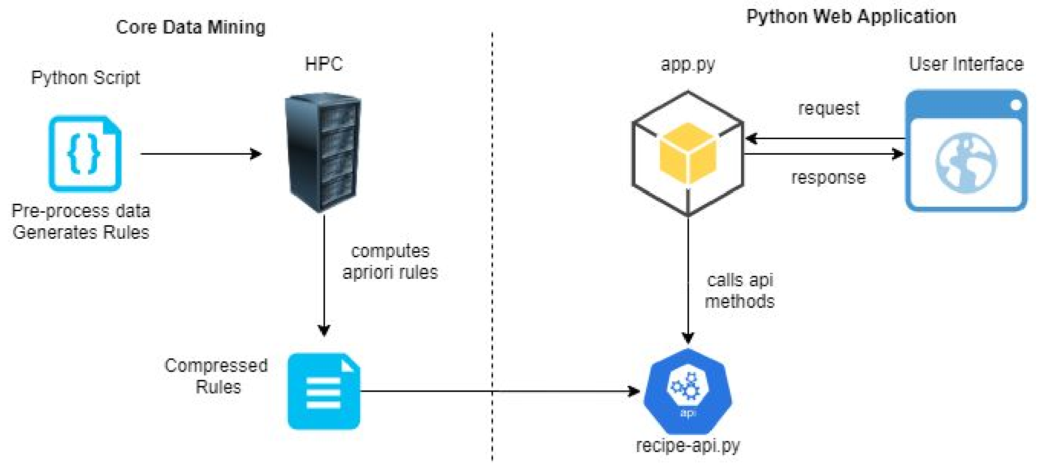


Fig 2: System Design

The second part is a Python Web Application created using Flask framework. It consists of three main files: `app.py`, `recipe-api.py`, `index.html` and `script.js`. The app.py is our back-end server which provides access to HTTP methods like `GET` and `POST`. The `recipe-api.py` provides all functionalities like getting unique ingredients and recommending new ingredients. The `index.html` is the HTML interface which the user interacts with and `script.js` provides the frontend main features, which are to invoke the `GET` and `POST` request and displaying the results into the UI.

USE CASE/GUI

The following steps are the general flow of the GUI, please refer to figure 3 below:

1. Users can add ingredient(s) that they have on hand, and our GUI will assist the user to select the ingredients based on our database.
2. [Extra Feature] If the user does not know what to add they can click on the 'Surprise Me' button. Our model will generate three random ingredients. The user can choose to keep clicking on the button until they find the ingredient they want to incorporate.
3. User can click on the 'Add' button to add the ingredient(s) to the current recipe
4. After adding one or more ingredients, the user can click on the 'Recommend' button and our model will run the recommendation system.
5. A list of recommended ingredients will be generated. Now, users can pick whichever ingredient they would like to add to their current recipe.
6. At the same time, an evaluation score will be generated on the bottom right box to measure the predictions at runtime.

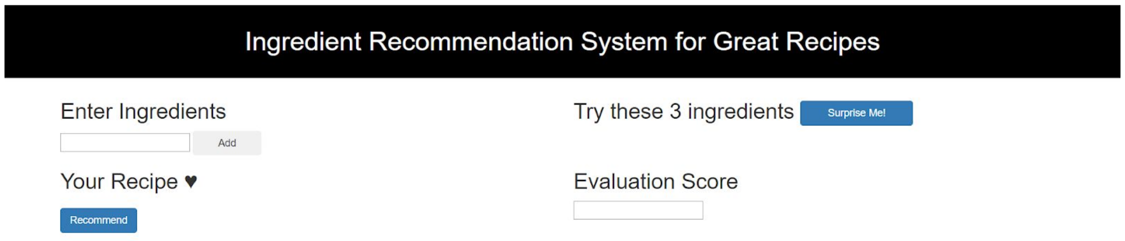


Fig 3: GUI

SECTION 3: EXPERIMENTS/PROOF OF EVALUATION

DATASET USED

Dataset	Link	Size	Type of Data
Yummly	https://www.kaggle.com/kaggle/recipe-ingredients-dataset	~40K (14.5 MB)	JSON

Epicurious	https://www.kaggle.com/hugodarwood/epirecipes	~20K (33.6 MB)	JSON
Datafinity	https://www.kaggle.com/datafiniti/food-ingredient-lists	~10K (5MB)	CSV
EightPortions	https://eightportions.com/datasets/Recipes/	~125K (195 MB)	JSON
TOTAL	https://dominikschmidt.xyz/simplified-recipes-1M/	~1.1M (62.2MB compressed)	NPZ

The data that we found was a combination of all of the datasets mentioned above. There was extensive preprocessing and cleaning that was done on the data, reducing the total number of unique ingredients from 1 million to 3500. The way the data was cleaned was mainly to reduce data sparsity. Those steps are:

- Some recipes contained measurements and verbs such as "`½ cup butter (unsalted), softened at room temperature`". Therefore, sizes/units, special characters, instructions, and other descriptions are removed.
- After those are removed, the adjectives from each ingredients such as '`baking potatoes`' are split up to `[baking potatoes, potatoes]`
- As a result, there are numerous dataset errors such as `olive oil` which mapped to `[olive, oil, olive oil]`, and `prepared yellow` mustard `[prepared, yellow, prepared yellow mustard, mutard]`. So, we had to remove them manually by searching for adjectives and other descriptions of an ingredient to make better recommendations.

METHODOLOGY FOLLOWED

Our project used the Leave-One-Out cross validation for splitting our main data set into a test set and a training set. The Leave-One-Out algorithm will randomly take out one ingredient from each recipe and add them to the test set (this test set will be our ground truth).

In terms of the evaluation of our proof of concept, we will be using the AP and ARHR metric to evaluate our methodology. AP (Average Precision) is a ranked precision metric that places emphasis on highly ranked correct predictions (hits), which is determined after each successful prediction. ARHR (Average Reciprocal Hit Rank) is similar to AP, but it weights the contribution of each item based on its inverse position to the rank list. However, it is important to note that ARHR only cares about the single highest-ranked relevant item, this does not mean that having a score closer to 1 is always better. Here is how we will evaluate the recommendation using those metrics: Once the user enters the ingredient(s) and receives the top-k recommended ingredient(s), we take those ingredients, sort them based on confidence values from highest to lowest. Given the ingredients that the user enter, we will search the

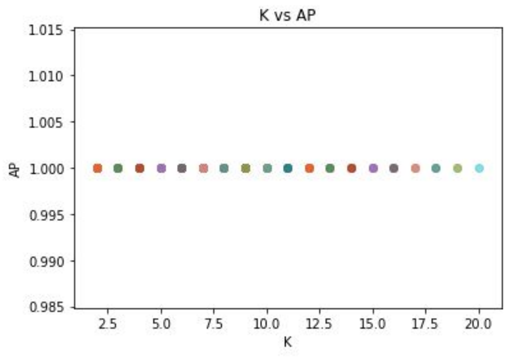
Rank	Ingredient	Hit
1	Onion	Yes
2	Chicken	No
3	Tomato	Yes

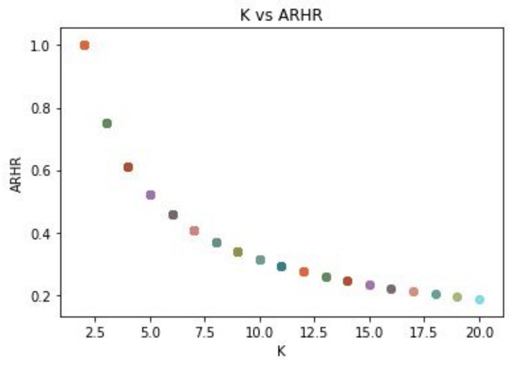
training set for any recipes that match with the combination of those. If any recommendations match with the ground truth, then it is a hit. Based on the number of hits, we calculate AP and ARHR and get the evaluation of the model. Referring to an example table on the left, we can calculate the AP and ARHR as follows:
Average Precision = ½ (1 + ⅔) = 0.83
Average Reciprocal Hit Rate = ½ (1 + ⅓) = 0.66

The main parameters that affect the rules generated by Apriori are the *minsup* and *minconf* values. We did not perform any specific evaluation on these two parameters. We generally want low values for both parameters (as close to 0 as possible) because we want to generate all the possible rules. Therefore, we used a low enough amount that the HPC was able to compute on a timely manner. We tried putting 0.0001, the job timed out at a limit of 24 hours. We tried putting 0.0005, the job timed out at a limit of 15 hours, both ran at 32GB. Finally, we put 0.005 for both *minsup* and *minconf* and was able to complete within 3 hours at 32GB.

An important parameter that we are considering is the number of ingredients to recommend. We have to find the optimal k-ingredients to recommend to give us a good evaluation score. If we recommend too few ingredients, there is a 50/50 chance that it would be a hit. If we recommend too many ingredients, then there might be a chance that we recommend the least relevant ingredient. The analysis below is to find the optimal k-ingredients to recommend.

GRAPHS





ANALYSIS

To generate the graph for AP vs K and ARHR vs K, we used 3 random ingredients as a sample and then chose different values of k-recommendations for each ingredient. The graph above shows how the score changes/stays as we increase the value of k-recommended ingredients. Here are the results of our samples for the last k = 20.

Ingredient	**Number of times the ingredient is in the rule's LHS**	**ARHR Score**	**AP Score**
bacon	24	0.15733159073972944	1.0

almonds	6	0.40833333333333327	1.0
salt	1627	0.0048998254792347335	1.0

The calculation of Average Precision and Average Reciprocal Hit Rate is given by these formulas:
AP = summation/hits
Where, summation = summation + (hits + 1) / k
hits = Number of hits
ARHR = summation/hits
Where, summation =summation + 1.0 / k
hits = Number of hits

In the case of AP vs K, we can see from the graph that AP is always equal to 1 no matter the value of K for all our tested examples, which are {bacon}, {almonds}, and {salt}. This means that the recommended ingredients always appear on the ground truth and keep showing hits on the evaluation table. We think it is because we set our support and confidence close to 0 and it allowed the algorithm to generate as many rules as it can. In the case of ARHR vs K, the ARHR is logarithmically decreasing as we increase the value of K for all our tested examples, which are {bacon}, {almond}, and {salt}. This implies that we are always getting the top hit regardless how many items we recommend. A lower ARHR does not mean that it is worse; for each increasing k, if all of the k-recommended items are hit, the ARHR score will be lower.

SECTION 4: DISCUSSION AND CONCLUSION

DECISIONS MADE

- [dataset] Considering the data set to choose was one of the major decisions we made. We picked up datasets, which provide us with a large number of recipes, and a wide variety of ingredients
- [preprocessing] We decided to remove some data errors because there were many ingredients that are repetitious and incorrect, for example `black pepper, ground black pepper, freshly ground black pepper, pepper, ground`. Even though it reduces the number of ingredients, it gives us better recommendation results.
- [algorithm] Another major decision is that we chose to implement Association Rule Mining using Efficient Apriori Algorithm instead of FP-Growth. Using Apriori, we are able to predict the next recommended recipe from just 1 ingredient, whereas FP-Growth requires pre-selected association rules.
- [evaluation] To evaluate our results, we used Average Precision and Average Reciprocal Rank Hit.

DIFFICULTIES FACED

- We were going to use nDCG as an evaluation metric but had difficulties on using it since it requires a similarity matrix of the ingredients. We consulted our Professor and he suggested using AP & ARHR, which worked great.
- At first we were struggling on getting the ground truth, to apply the evaluation metrics after consulting with our Professor, he recommended using Leave-One-Out and it worked really well on our model.
- It was difficult to send and receive multiple huge files (training set) using the API. So we had to preprocess the association rules before the GUI started.
- Integration between Python and Javascript was difficult and took a very long time. Even though we are using a framework, we all had few experience using Javascript and had to learn a lot of things from scratch.
- We wanted to generate all the rules with 0 support but the HPC timed out when we used 32GB, 24 hour limit, and 0.0005 *minsup* and *minconf*. So, we had to settle with 0.005.

THINGS THAT WORKED

- Removing the repetitions and bad words greatly improve our recommendation in terms of user-friendliness. We avoided being recommended items such as `ground`, `prepared`, `sliced`, `softened`.
- During testing, we were able to generate recommendations with 1.0 AP score, which means that the Apriori algorithm worked well for our project.
- We were able to integrate our JavaScript UI code with the Python backend so as to take the user's starting ingredients and then suggest him the top k ingredients and also suggest some random ingredients if the user clicks on the surprise me button.
- The Average Precision and Average Reciprocal Hit Rate calculated from the recommended ingredient showed us the accuracy of the recommendations.

THINGS THAT DIDN'T WORK WELL

- We wanted to create a recommendation system that recommends serendipitous ingredients, however given the nature of it, there was no accurate way to measure those recommendations.

CONCLUSION

We utilized concepts in data mining in hopes to enhance day-to-day lives, in this case to help home cooks create great meals. We looked at multiple datasets from websites and repositories and our team was able to collect data that is large enough to generate the association rules. We also explored classic recommendation algorithms and found out that Association Rule Mining works best for our project given the absence of user preference. The Association Rule Mining that we performed through the Apriori algorithm was able to provide relevant ingredient recommendations. This is proven by the evaluation metrics that we are using, which are AP and ARHR. We are able to get perfect and close to perfect AP and ARHR scores for our recommendation system. Therefore, we are proud to present a working prototype of our ingredient recommendation system! We hope to have a better version deployed in the future so homecooks can actually use this as an application and take advantage of our model. In the future, we plan to do improvements such as having a larger dataset and better preprocessing steps, be able to generate rules with 0 support (a more advanced HPC), a GUI with better aesthetics, and more portable code.

SECTION 5: PROJECT PLAN/TASK DISTRIBUTION

Name	Technical Task	Non-Technical Task/Housekeeping
Gaurav Karki	- Completed Integration of GUI Frontend with Python Backend using Flask - Wrote JS/HTML/CSS - System design see Fig 2 - Implemented ARHR/AP - Implemented final recommendation	- Worked on Use Case, System Design part of the Report
Mega Putra	- Wrote 100% API for backend - Suggested libraries for main Python file e.g pickle, collections - Suggested to further preprocess data to avoid bad recommendations - Assisted in Implementing ARHR/AP by explaining how to use the ground truth from user input	- Discovered the main dataset - Setting up all meetings - Running files to HPC and provide updates to group - Completed 100% Powerpoint presentation slides - Completed 50%+ of report, responsible for proofreading
Mayank Verma	- Preprocessed the data by removing the list of bad words and empty recipes. - Researched the optimum number of k recipes to recommend that gives best evaluation score by generating the AP/ARHR graphs Vs K-recommended ingredients - Analysis of the accuracy graphs and results	- Worked on the final report

Other comments:

- During the last 4 weeks of the quarter, our group meets 2-3 times per week to complete the project.
- Each group member responsible for a task was able to complete their respective task accurately and in a timely fashion.