Recipe Recommendations

Easton Potokar

March 2020

Abstract

We seek to explore making a personalized recipe recommendation system based on ingredients, tags, and past user reviews. We analyze both the accuracy of the recommendations as well as the temporal complexity of making them. We find that we can make very accurate recommendations to satisfy what cravings people may have and those they didn't know they had.

1 Problem Statement and Motivation

Machine Learning algorithms that make recommendations to users are becoming more and more common and are seen all over the internet for movies, friends, web searches, etc. A less common application is for recipe suggestions. Is it possible to find a perfect recipe for someone given their past preferences of foods? Can we do it in fast enough so we don't lose their interest?

This has been attempted using matrix factorization [1], which essentially utilizes both components of an SVD decomposition. There has also been work done to use a hybrid approach that utilizes both the reviews of users and the attributes of the recipes, whereas more traditional methods use one or the other [2]. We seek to continue along this path of more traditional methods, while also analyzing the speed of our algorithms to see how viable they would be in production.

2 Data

Our data was gathered from [3], whose dataset can be found here. The data consists of more than 180 thousand recipes (made up of their ingredients, calorie level, and various tags describing them), and 700 thousand user reviews of the 180 thousand recipes all scraped from Food.com and ranging from 1 to 5 stars. The original authors of the data used various natural language processing techniques to parse the recipes into about 8 thousand unique ingredients, around 500 unique tags (both characterized with a 1 when they are present, 0 when they aren't), and a calorie level that is a 0, 1, or 2 denoting low-calorie to high-calorie recipes. They did this to create a personalized recipe generation system, similar to our task here. To visualize

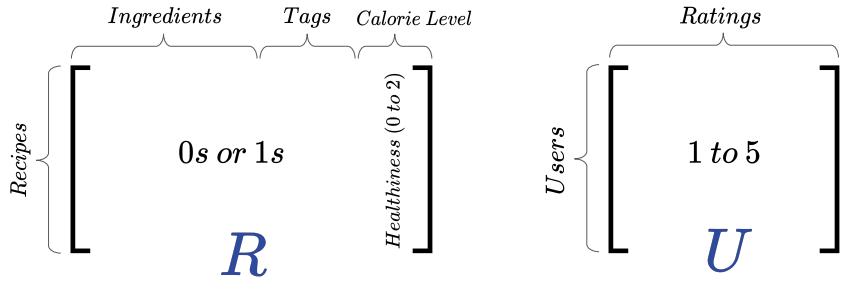

Figure 1: Data Overview

what we're saying, see Figure 1. We denote the recipe dataset as R and the user dataset as U throughout the rest of the paper.

The data appears to be very reliable, having been scraped directly from Food.com with the scraper being publicly available on github [4]. Furthermore, it was used in a published research article, so one would hope everything was done ethically and correctly. The dataset is plenty large enough to give use meaningful answers to our questions and should be able to make a fairly robust recommendation system.

3 Ethical Ramifications

As with all recommendation systems, we must be careful of what sort of bias we program into our algorithms. It would be very easy to heavily weight the algorithms to favor low-calorie recipes (healthier options) in an effort to push the public towards eating more healthy. While this would likely be well intentioned, in a way it is removing one's freedom of options, and is manipulating users' behavior. Many similar biases must also be carefully considered, whether it be toward a certain ingredient/brand (company sponsorship), a certain ethnic preference (potentially racist), etc.

4 Methods

4.1 Data Preparation

Our data is on the format as seen in Figure 1. This is a far from perfect solution. Note that the importance of a recipe having salt will be just as important as a recipe having jasmine or a jalapeno pepper. Commonly used in document classification, we use "TF-IDF" (term frequency–inverse document frequency) to help with this problem. In our case, we change each entry using the conversion (where item means either ingredient/tag):

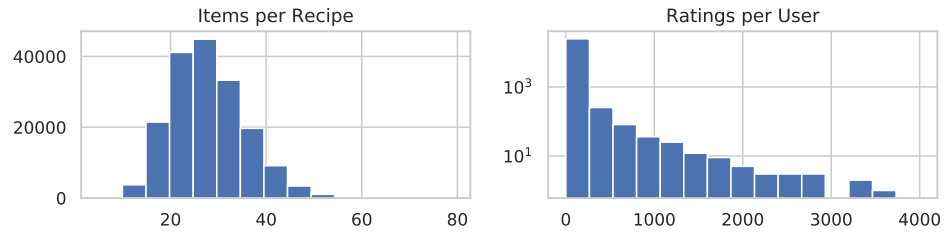

Figure 2: Data Statistics

$$\frac{\text{Quantity of } i\text{th item}}{\text{Total \# of items in Recipe}} * \log\left(\frac{\text{Total \# of Recipes}}{\text{\# of Recipes with } i\text{th item}}\right)$$

We perform the same transformation on the user dataset, replacing item with rating, and recipes with users. Throughout this document we analyze our data in both forms in order to visualize which data format helps the algorithms perform the best. We denoted these datasets as $Rhat$ and $Uhat$ as compared to R and U for the original datasets.

To give ourselves a little intuition, we also visualize the distributions of how many items a recipe has and how many recipes a user has rated. This can be seen in Figure 2.

4.2 Recommendation Process

Most recommendation systems or "filtering" systems are considered either content-based or collaborative-based filtering. Content-based systems make recommendations on attributes of the items; in our case, this would be ingredients and tags of our recipes. Collaborative-based filtering uses the history of other users who are similar to us to recommend items. We'll explore using both. Our workflows will be shown in Figure 3.

4.2.1 Collaborative-Based Filtering

It's here that we'll use our new algorithm, density-based spatial clustering of applications with noise (DBSCAN). It has two hyperparameters, ϵ and min_samples. It begins with a random point and checks if there is min_samples within ϵ of it. If so, all these samples are added to it's cluster. It then checks the same thing for those samples, and iteratively continues checking all points in the cluster. Once there is no more samples to check in the cluster, randomly choose another point and perform the same to create another cluster. This allows DBSCAN to detect non-convex clusters, unlike KMeans and GMM, but still be relatively faster that MinCut and t-SNE.

We'll go over what was attempted in each step of collaborative-based filtering.

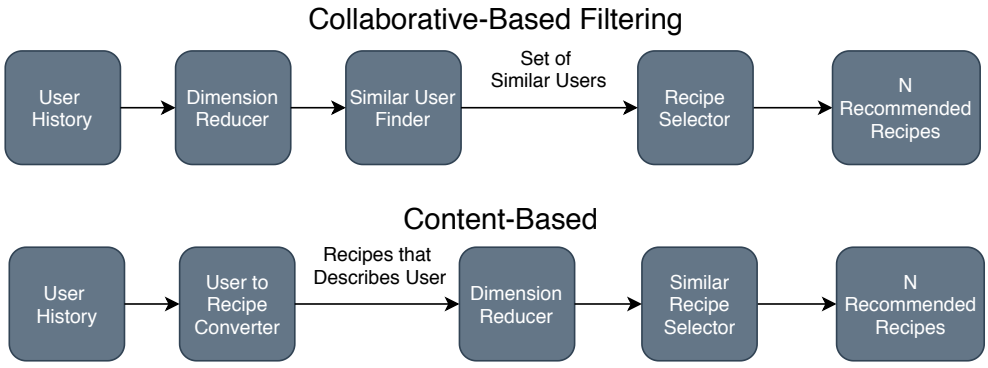

Figure 3: Recommendation Systems Flow

1. **Dimension Reducer**. PCA, LDA, NMF and Kernelized PCA were all used. Probablistic PCA and Sparse PCA were also attempted, but don't play well with sparse input. Kernelized PCA was also too computationally expensive and often abandoned.

2. **User Finder**. We tried various algorithms for this:

   - k Nearest Neighbor (kNN).
   - Various Clustering Algorithms. Return all other users found in same cluster. Attempted using KMeans, GMM, MinCut, and DBSCAN.
   - Combination of clustering and kNN. Return k nearest neighbors found in cluster. Could potentially be more computationally effective.

   We also take care to make sure our user finder doesn't return the input user.

3. **Recipe Selector**. Taking the users passed to it, count number of times recipes are rated by user set along with their rating. Choose from among recipes by:

   - Sorting by sum of ratings and return largest N sums.
   - Sorting by frequency and then by average rating. Return N largest.
   - Sorting by average rating and then frequency. Return N largest.

Note if our algorithm returns too few recommendations, (ie a cluster ends up being a singleton or something of the sort), we simply fill in the rest with random recipes.

4.2.2 Content-Based Filtering

We'll go over what was attempted in each step of content-based filtering.

1. **User to Recipe Converter**. We attempt this in two ways:
    - Take all recipes with rating from user above a threshold. This method keeps each recipe separate (SR).
    - Take all ingredients from recipes with rating from user above a threshold and combine into one huge recipe. (TR)
2. **Dimension Reducer**. Dimensions were reduced in the same way as in the Collaborative-Based Approach.
3. **Similar Recipe Selector**. In the case where the set of user describing recipes is a singleton, simply take the N NNs. In the case it isn't, take k NNs of each recipe and return the N most frequent.

4.2.3 Scoring

To evaluate our models, we removed around 100 thousand reviews from random users. We will then take in the user as input and try to recommend the removed recipe that we know they rated or a recipe similar to it.

To quantitatively define how good a prediction is, we use two different loss functions. Let R be the recommended recipe, G the goal recipe and I_R, I_G be the set of items that each has. We define our loss functions as:

$$Int(R, G) = |I_R \cap I_G| \qquad Com(R, G) = -|(I_R \cap I_G)^C| \tag{1}$$

We desire to maximize both. $Int(R, G)$ rewards us for correctly recommending items; $Com(R, G)$ punishes us for recommending incorrect items. For example, sometimes we may recommend something with cheddar cheese instead of American cheese and $Com(R, G)$ punishes that, even though it's likely close enough. Similarly, $Int(R, G)$ rewards recipes that have many ingredients. Using both gives us more insight on how our algorithms perform.

To get our total loss, we simply take the mean of all N (we have arbitrarily chosen N=5 for our tests) recommendations and then the mean of all the data points together.

5 Results

We ran a grid search over each combination of dimension reducer and similarity finder to find the best hyperparameters for each combination of the two. We also timed both the fitting as well as the prediction steps to be able to see which methods are actually feasible in practice. We did this separately for both the intersection and the complement metric to see how each could be maximized. The results can be seen in Figures 4, 5 and 6.

After a few test were done, we found that the best recipe selector for collaborative-based filtering was sorting by sum of ratings, almost uniformly. To help narrow down our search, we used it exclusively.

Certain combinations had to be abandoned as well due to high processing requirements that just didn't make them feasible in practice. This includes kernelized PCA with content-based filtering, as well as using MinCut along with NN.

As a baseline, we tested the performance of randomly predicting N recipes and recommending the N most popular recipes. Random recipes got 7.66 and -39.12 and popular recipes got 9.58 and -42.515, in the intersection and complement metrics respectively.

We also include the best results in each category in Figure 6. We calculated best overall by averaging their rank in each metric.

6 Analysis

6.1 Collaborative-Based Approach

First note that nearly none of these algorithms were particularly good in the complement metric. This means that on average, no particular algorithm was extremely

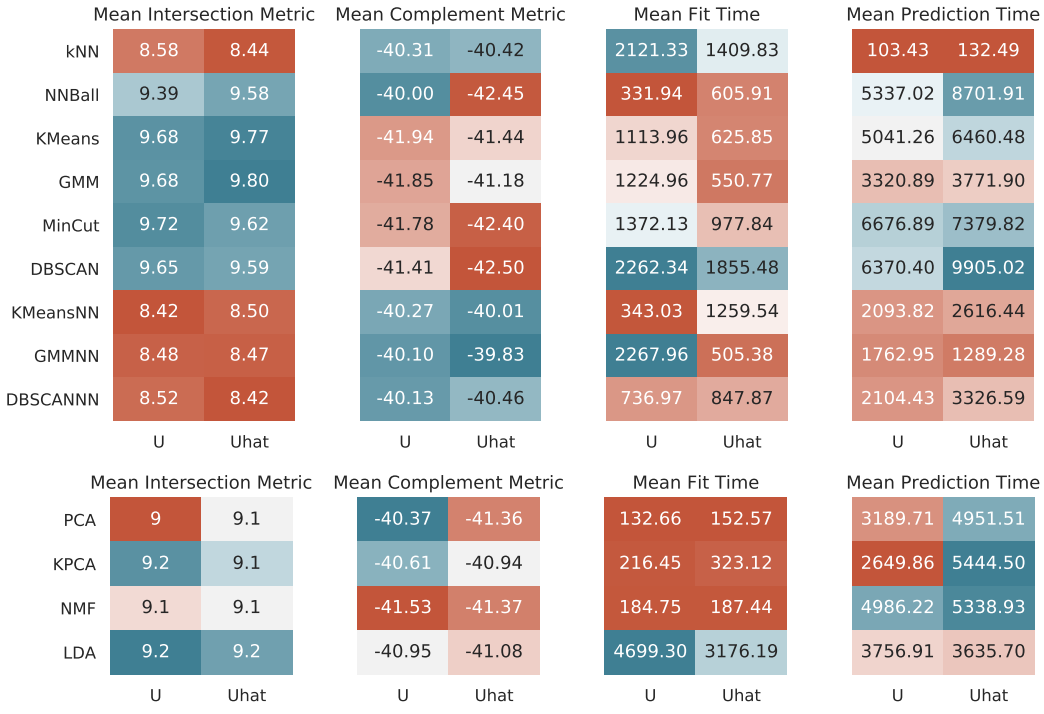

Figure 4: Comparison of Collaboration-Based Recommendation Algorithms

adept at the complement metric under all circumstances, but doesn't mean that a certain combination won't perform extremely well.

Also, the clustering algorithms all did almost uniformly well, with very little separating them. This surprises me since I expected the clusters to be relatively non-convex, but based on these results, I would actually assume that most of the resulting clusters are convex. They performed a little better than recommending popular recipes, and I expect would bring better variety to users as well, making them particularly worth having around. It also appears the shortcomings of kNN also fell upon the combo cluster and kNN algorithms as they performed better than random, but still worse than just recommending popular algorithms. I would also guess that NNBall essentially acted as a "local" cluster by just selecting everything within a radius of itself, thereby circumventing the shortcomings of kNN (which may reach very far away to select a neighbor).

Also, the choice of data, original or TF-IDF, made very minimal difference in the results, which is extremely surprising. This could be due to most users having rated so few recipes out of the total amount of recipes that it ends up looking like they're close/not close regardless of the data.

Note that the choice of dimension reducer had little to no effect on average either. Again, this doesn't mean a certain combination of dimension reducer and similar user selector might be dynamite, but as a whole, it didn't matter too much.

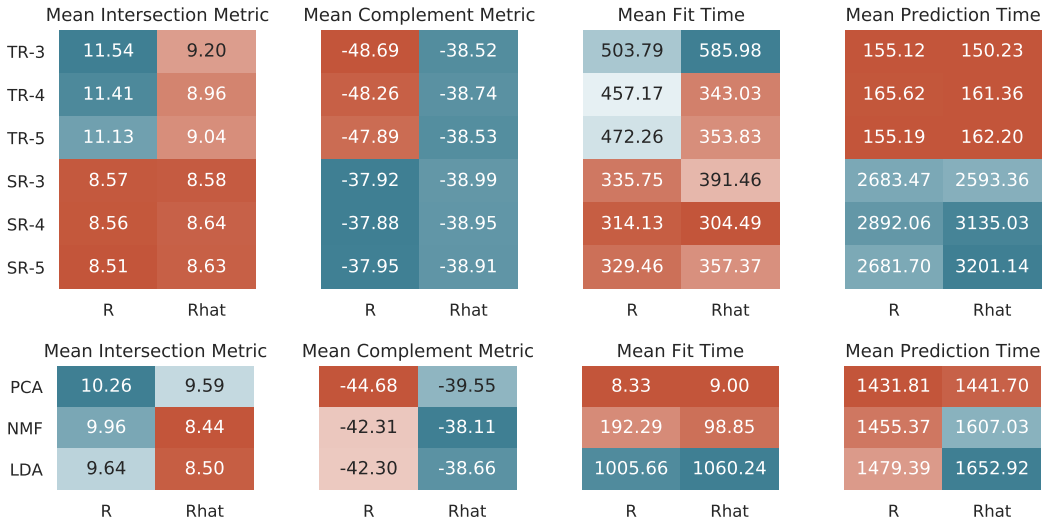

Figure 5: Comparison of Content-Based Recommendation Algorithms

6.2 Content-Based Approach

Right off the bat, it can be seen that mashing all previously liked ingredients of a user into a recipe and finding similar ones (TR) likely just sought out recipes that have many, many ingredients, which means it should probably be avoided. On the other hand, separating out the recipes had the opposite effect, performing extremely well in the complement metric and poorly in the intersection metric.

Once again, the choice of dimension reducers appears to mostly be a wash.

Finally, the choice of data in this scenario did play a huge role in the results. Notice that the use of TF-IDF really improved performance in the complement metric, possibly playing a pretty active role in transforming the vector space.

7 Conclusion

Recommendations systems are common all over the web from Netflix to eBay. One space where the usefulness of these hasn't been researched extensively is the food recipe industry. In our analysis, we found no clear-cut perfect algorithm for all scenarios. Depending on what you what your users to experience - either extremely similar recipes, or a little more freedom with some overlap in previous history - the choice of machine learning algorithm is going to play a large role.

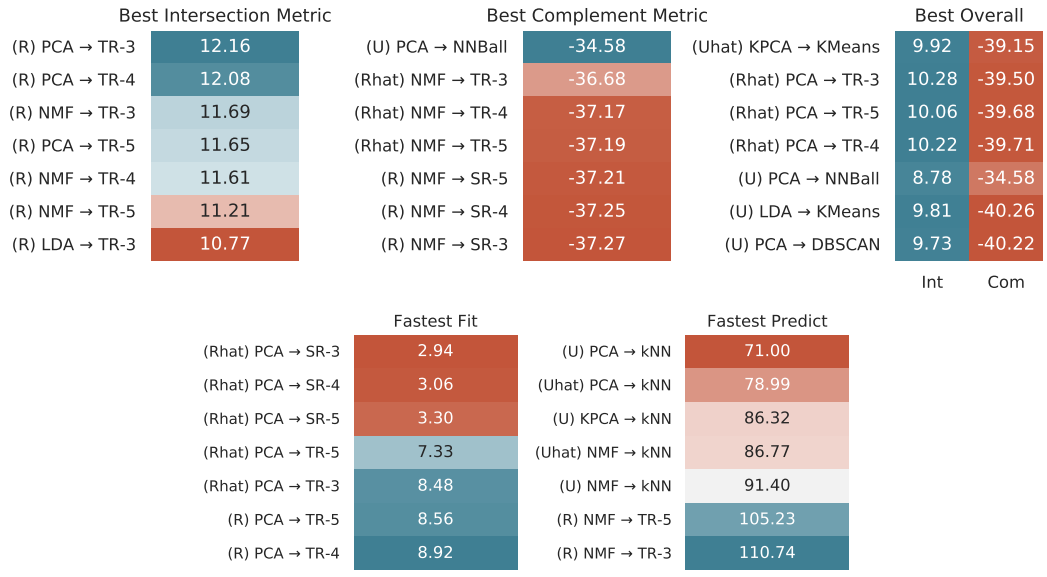

Figure 6: Best Results

References

[1] Sai Bharath Goda. Recommender system for recipes. Master's thesis, Kansas State University, 5 2014.

[2] Anirudh Jagithyala. Recommending recipes based on ingredients and user reviews. Master's thesis, Kansas State University, 5 2014.

[3] Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. Generating personalized recipes from historical user preferences, 2019.

[4] Bodhisattwa Prasad Majumder, Shuyang Li, Jianmo Ni, and Julian McAuley. Recipe personalization. https://github.com/majumderb/recipe-personalization, 2019.