Merging Collaborative & Content Filtering for Recipe Recommendations

Jason Summer
May 2020

Executive Summary

In the saturated market of recipe aggregation, organizations seeking to grow loyalty and expand their user base should consider machine learning techniques to simplify the customer experience. There is no shortage of quality recipes digitally available. However, simplifying the process to find these recipes and ensuring consistency of these quality recipes is key.

The following analysis details how collaborative filtering merged with content filtering can be used to drive loyalty of an existing customer base. Furthermore, it details the effect of a saturated market on usership and how providing optimal first recommendations is crucial for growth.

Proposal & Overview

From allrecipes.com to the Mealime app, recreational chefs today have an overwhelming digital arsenal of ways to find recipes. In the past, peer reviews were only possible by polling the audience at your dinner party. Now however, recipe reviews are a core part of online recipe aggregators. Furthermore, some recipe aggregators are complemented by interactive grocery lists. Mealime's Meal Plans and Recipes app is the perfect example - selecting from a relatively small library of recipes, one can create a meal plan and receive an automatic grocery list based on the selected recipes' ingredients. As of April 2020, the app was rated 4.8 out of 5 stars with 32.1K user reviews.

What is missing from websites like allrecipes.com and apps like Mealime, is bridging recipe recommendations with large scale recipe aggregation coupled with grocery list generation. Specifically, by creating a network of recipes, users, and reviews on par with volumes seen at food.com or allrecipes.com, a highly personalized recipe recommendation engine could be created. Furthermore, ingredient lists from recipe recommendations could be created further filtered by user selection or input, such as ingredients already available on hand. While many companies currently have 1-2 of these 3 key features (recipe aggregation, recipe recommendations, and grocery list management) no single offering combines them in a seamless manner. Thus, such a concept could be of interest to both existing entities and Food & Drink industry newcomers.

In an effort to construct a proof of concept prototype, data from https://www.kaggle.com/shuyangli94/food-com-recipes-and-user-interactions#RAW_interactions.csv was leveraged. The data consist of 180K+ recipes and 700K+ recipe reviews covering 18 years of user interactions and uploads on Food.com. Using machine learning, the engine considers both content filtering (based on recipes' keyword tagging) and collaborative filtering (based on similar chefs' reviews). In addition to providing a list of recipe recommendations, key features also include shopping list creation, relating recommendations to similar recipes from a user's rating history, and providing the most popular recipes to new users.

Approach and Methodology

Each combination of a user and a recipe has a rating from 0 to 5, with 5 being the best possible review. Thus, recommendations to users will provide the recipes with the highest predicted rating using collaborative filtering based on similar users' ratings. Multiple algorithms from the Surprise package will be evaluated in their overall prediction error.

Additionally, recipes will be compared pairwise for similarity to provide historical context to users through content filtering. Specifically, any recipe recommendations that share a degree of similarity with a user's previous recipes will be denoted as such to help users better compare their recommendations.

Deliverables

The final deliverable package consists of this corresponding documentation, a Jupyter Notebook, and a brief PowerPoint presentation. The Jupyter Notebook features instructional walkthroughs and corresponding code to be executed at each step in the analysis and model creation, from data ingestion to model output and assessment. The PowerPoint presentation provides a high level overview of the opportunity identified, analysis, recommendation methodology and review, and feature engineering.

Data Ingestion and Wrangling

To begin, two csv files were imported. The first file, which will drive the collaborative filtering effort, was the interactions file. It quite simply features a user, recipe and corresponding rating information.

Field	Description
user_id	food.com user identifier that submitted rating for corresponding recipe
recipe_id	food.com recipe identifier to match to corresponding recipe in recipe file

date	date of review
rating	0-5 rating
review	user entered free-text pertaining to the rating submission

The other file included in the analysis details recipe information. Specifically, each record corresponds to a recipe referenced in the previous file's recipe_id field. The recipe file is rich with additional information pertaining to cook time, contributor, recipe tags, nutrition, instructions, ingredients, etc. The tag field was used for content filtering.

Field	Description
name	recipe name
recipe_id	food.com recipe identifier
minutes	number of minutes to prepare recipe
contributor_id	1-5 rating
submitted	user entered free-text pertaining to the rating submission
tags	recipe keywords
nutrition	nutrition information (calories (#), total fat (PDV), sugar (PDV) , sodium (PDV) , protein (PDV), saturated fat (PDV) , and carbohydrates (PDV))
n_steps	number of steps in recipe
steps	text for recipe steps, in order
description	user-provided description
ingredients	list of ingredient names
n_ingredients	number of ingredients in recipe

Following data import, minor data cleaning was necessary. A single recipe record did not have a corresponding name and would thus be difficult to recommend to a user. The unnamed recipe record was removed from both the recipe and ratings information.

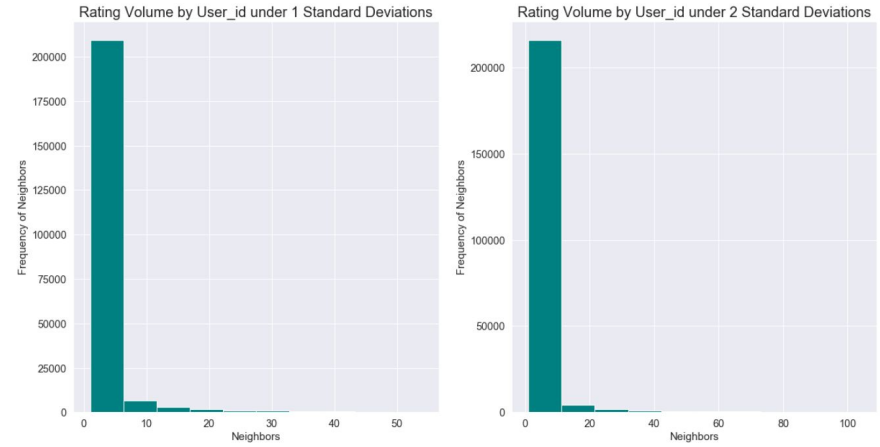
In the next section, the ratings were treated briefly as a bipartite network structure for visualization and exploration. To properly create bipartite graphs, the two bipartites cannot share unique identifiers. Thus, a new user id and new recipe id were created using prefixes of 'u' and 'r' respectively.

Exploration and Analysis

Rating Submission Volume

Review of the ratings and recipe information reveals that there are 226,569 unique users and 231,636 unique recipes. Furthermore, the average number of rating submissions per user is ~5 while the median is 1. The per user submissions ranges from 1 to 7,671. More quantitative information of the distribution is below for reference.

Rating Submissions by User	
count	226,569
mean	5
standard deviation	50
min	1
25% percentile	1
50% percentile	1
75% percentile	2
max	7,671

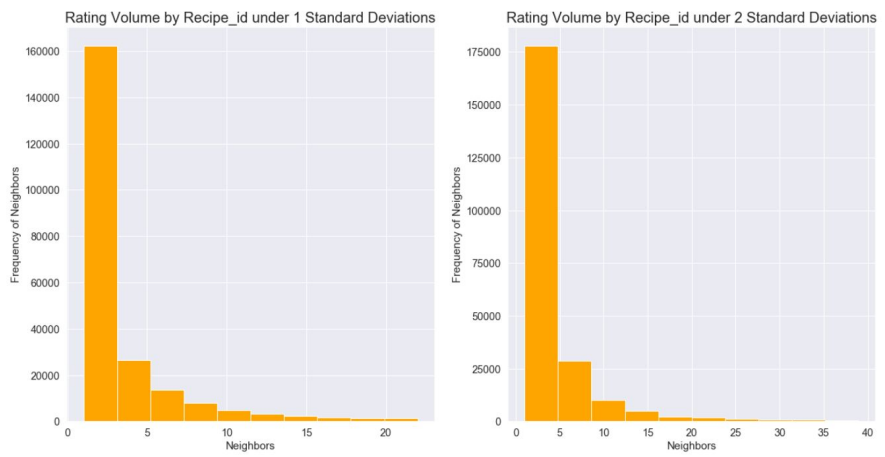


The distribution highlights some key details and considerations for recommendation construction. Specifically, there are no users without ratings submitted. Thus, this particular dataset does not provide an opportunity to address cold start users, otherwise known as users who have not submitted ratings. For these cold start users, recipes that have the highest average (or median) rating and have beyond a threshold of rating submissions should be recommended in descending order of their average (or median) rating. In essence, the most popular recipes should be recommended to new users.

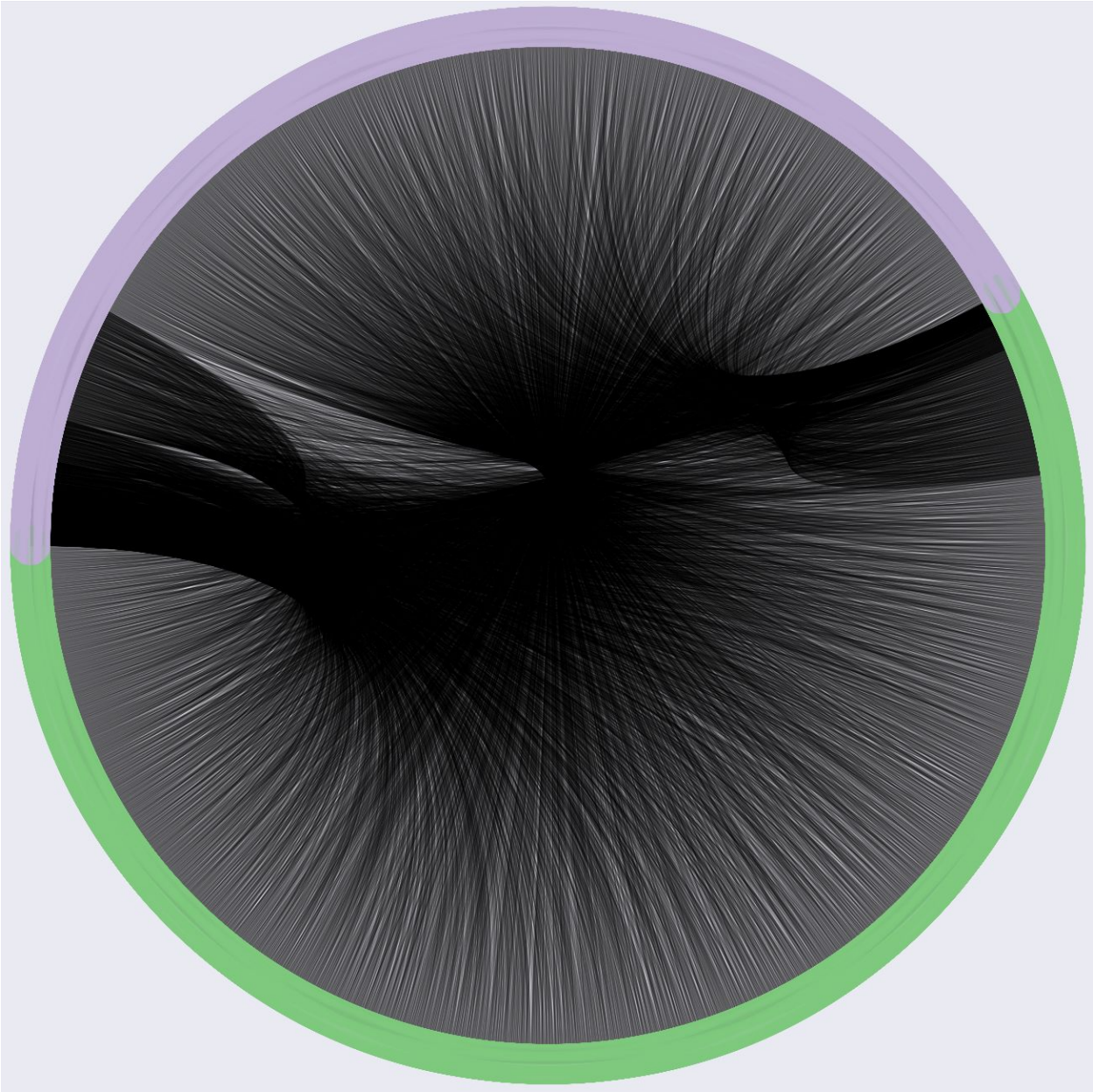
The contrast of the mean and median above also illustrates that the vast majority of users have submitted only 1-2 ratings but there are a smaller proportion of users on the much higher end of rating submissions thus skewing the mean away from the median. There is a risk with the majority of users only submitting 2 or less reviews that similarities and collaborative recommendations could be less strong than for a pool of more active users.

With ratings and users nearly 1-to-1 in total volume, the distribution of rating submissions by recipe is very similar to the distribution by user shown above. The average ratings by recipe is again ~5 while the median is 2. The 75th percentile is higher at 4 ratings while the max is 1,613. Thus there is a less exaggerated long tail in the distribution of ratings by recipe.

Rating Submissions by Recipe	
count	231,636
mean	5
standard deviation	18
min	1
25% percentile	1
50% percentile	2
75% percentile	4
max	1,613

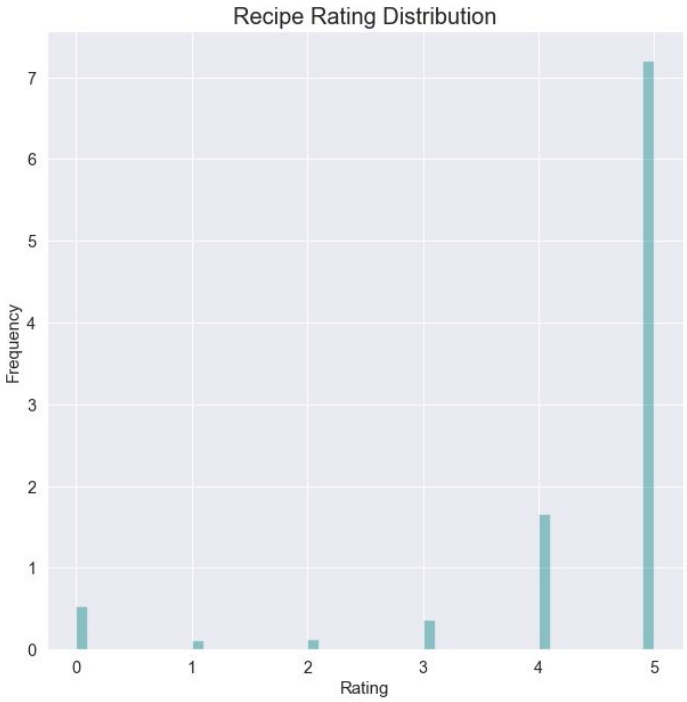


Based on the above distributions, there are 1,132,366 rating submissions. Due to the sheer volume of ratings, a single network graph does not lend itself for granular visualization. However, simply visualizing a random sample of 10,000 of a bipartite graph does show the sparse nature of the rating data. For each bipartite (recipes and users) there is a small proportion of darker density (on the left and right sides) while the vast majority is evenly spread into very thin associations. Again, most recipes and users are only involved in a few ratings at most.



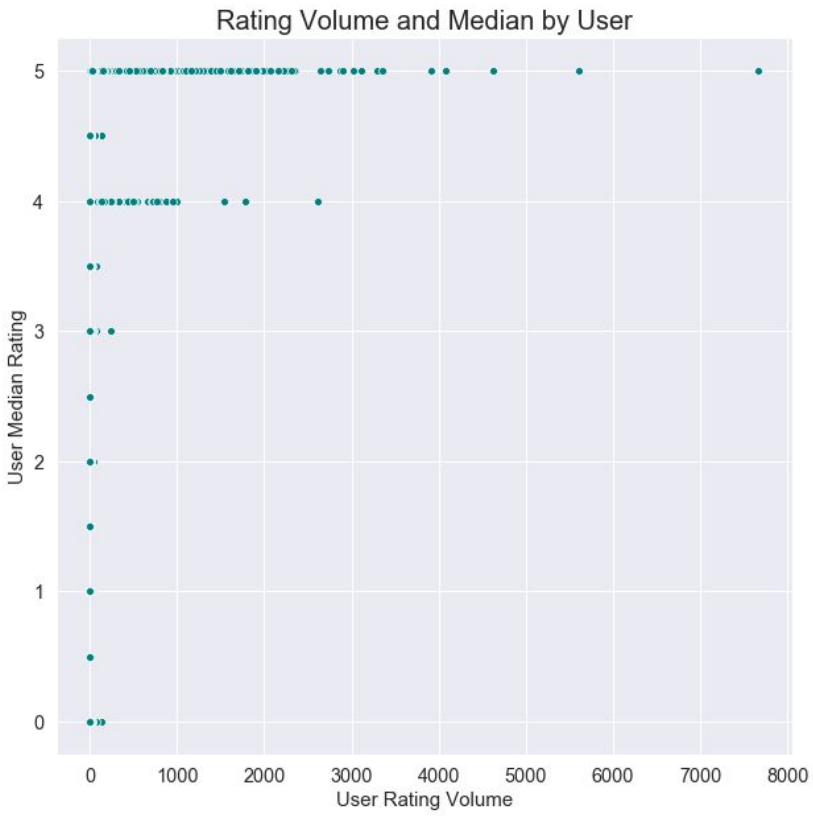
Rating Submission Scores

While users generally only submit a few ratings, these ratings seem to be strong, i.e. generally 4 or 5. Evaluating the distribution of common rating submissions shows a very obvious favoritism to submitting positive ratings.



Additionally, there are very few rating submissions of 1-3 and obvious rating submissions of 0. Food.com allows a review to be submitted without any stars specified, which indicates a rating submission of 0 as shown. Due to the obvious favoritism towards higher ratings, using a median instead of mean as a popularity assessment may be appropriate. Furthermore, since rating is the prediction target, this particular dataset is somewhat imbalanced, suggesting a need to also evaluate models using a more balanced dataset.

Comparing median rating with a user's volume of rating submissions also reveals a positive relationship. Specifically, users tend to submit higher ratings as they submit more ratings. As mentioned in the introduction, the recipe aggregation marketplace is quite saturated. This suggests that because users may only consider 1-2 recipes on a platform, a single "bad" recipe selection can cause very swift attrition. This state of competition and user behavior underscores the need for quality recommendations for both new and low usage customers.



Collaborative Filtering Modeling

Model Preparation

As mentioned above, any recipe recommendation should treat existing and new users differently. Recommendations for users that have previously submitted ratings should be primarily based on collaborative filtering. Alternatively, new users should be provided with the most popular recipes. To differentiate the two types of users, a function was created to split them. Furthermore, because the Surprise package used for collaborative filtering does not accept ratings of 0, these ratings were changed to 0.1 to retain the designation of a minimum rating.

Two additional functions were then created to transform the data into an acceptable format for the Surprise package. First, the rating submissions were reduced to featuring only user, recipe, and rating as a float and labeled as ‘estimator’. Then, using several methods of the Surprise package, a function was created to transform the ratings into a sparse-like matrix with columns corresponding to recipes and rows corresponding to users. The intersection of the two indicates the rating submitted.

Algorithm Selection

The Surprise package offers 11 out of the box algorithms for collaborative filtering including baseline algorithms, neighborhood-based methods, and matrix factorization approaches. Using the package's built-in cross-validation function, the 11 were compared using a random sample of 10,000 ratings and assessed based on the root mean squared error. Below are the results including time to train in the last column.

Algorithm	Mean RMSE	Training Time
SVD ++	**1.0533**	**2.197224**
KNN Baseline	**1.054044**	**0.734534**
SVD	**1.054203**	**0.990171**
Baseline Only	**1.054441**	**0.079275**
KNN Basic	1.060911	0.672592
KNN with Z-score	1.070732	0.967633
KNN with Means	1.070836	0.757333
Slope One	1.071442	1.372625
Co-Clustering	1.084257	2.124266
NMF	1.123129	2.692995
Normal Predictor	1.334555	0.022809

As illustrated in the chart, the top 4 algorithms were very close in performance. In fact, depending on the random seed, the top 4 oftentimes changed order. However, while SVD++ (an extension of SVD algorithm that accounts for implicit ratings) had the lowest error, it also had required over 2x the training as the other top performers. Furthermore, cross-validation examples from http://surpriselib.com/ indicate a training time over 9x any other algorithm offered in the package. As a result, SVD and Baseline were selected for hyperparameter tuning and more thorough train time assessments. SVD is a matrix factorization-based algorithm while the Baseline Only algorithm predicts baseline estimates for each given user and item.

Algorithm Tuning

Since a sample of 10,000 was used in the initial cross validation, time to train was re-assessed using SVD and Baseline Only with slightly larger samples. Evaluating the 2 algorithm's training

time on ~100k and ~300k samples shows that training time does not quite scale linearly with increased data size. For example, SVD train time on ~100k was 13.4 seconds and 1 min 36 seconds on ~300k rows. More importantly however, neither algorithm shows unreasonably high training times on ~300k rows. Thus, with a full dataset of just above a million records, both algorithms can be pursued with minor concern for training runtime. On the contrary, the SVD++ algorithm proved to be too memory intensive given resources at hand.

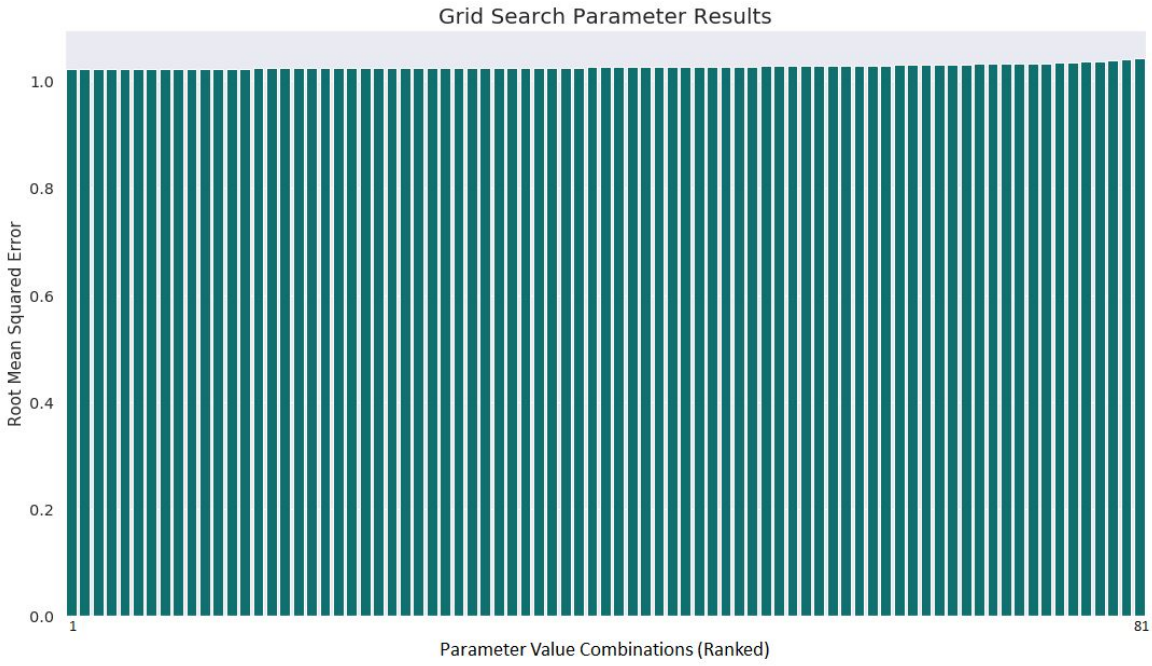
The SVD algorithm uses stochastic gradient descent and attempts to characterize the data using fewer dimensions than the raw data. Doing so offers 4 basic parameters for tuning.

- n_factors: number of latent features used to identify similarities in preferences
- n_epochs: number of iterations of the stochastic gradient descent used to minimize regularized squared error
- ir_all: learning rate for all parameters
- reg_all: regularization term for all parameters

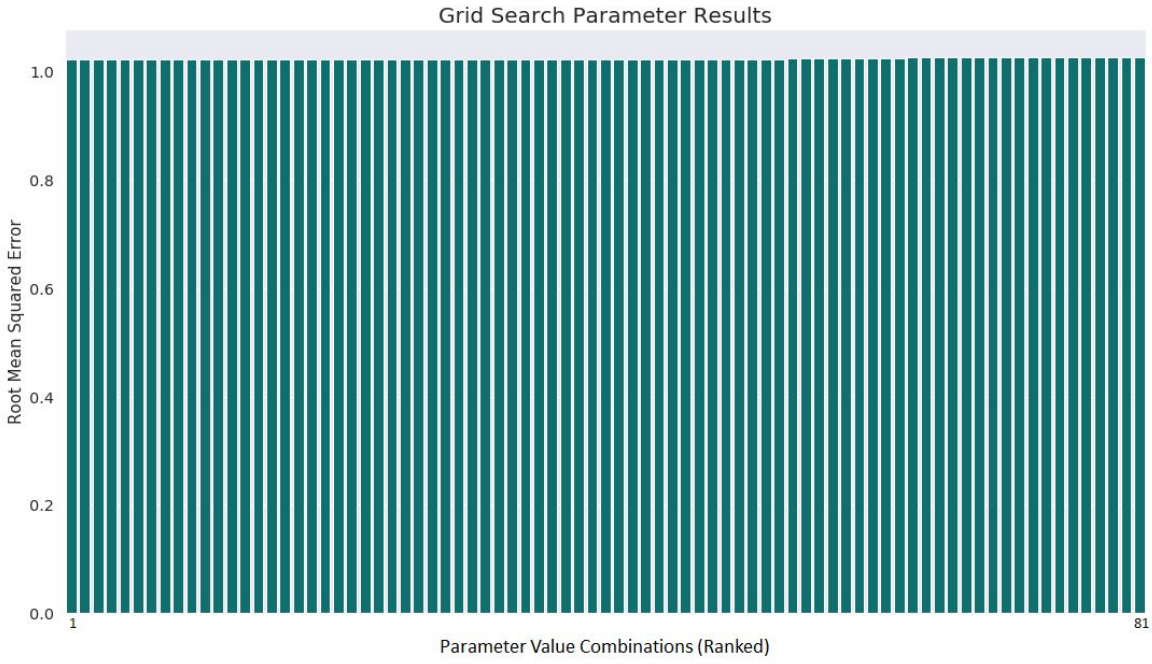
To begin, the below ranges were established for SVD's hyperparameters. Tuning was performed using a random sample of 100,000 ratings.

Hyperparamter	Range
n_factors	[**25**, 50,100]
n_epochs	[20, **30**, 50]
ir_all	[**0.00**2, 0.005, 0.01]
reg_all	[0.02, **0.1**, 0.4]

Again using root mean squared error the optimal combination of the above parameters was 25, 30, 0.002, and 0.1, respectively (highlighted above). However, worth noting is that tuning does not show significant changes in performance except for a handful of combinations at the right end of the below chart.



While significant improvement through more tuning is not guaranteed to show increased performance, it's a worthy exercise since rather large ranges were initially used. Thus using the optimal parameter values from above as middle points of more narrow ranges, the below shows the rather flattened performance of the second cross-validation.



Below are the optimal values selected for the SVD algorithm.

Hyperparamter	Optimal Value

n_factors	25
n_epochs	30
ir_all	0.003
reg_all	0.1

The Baseline algorithm offers two alternative methods to estimate and reduce the regularized squared error of each user and recipe combination. These two methods serve as the hyperparameter values that were evaluated.

- sgd: stochastic gradient descent
- als: alternating least squares

Using the same random sample in the prior cross-validation effort, the sgd and als methods of the Baseline Only algorithm were cross-validated. The stochastic gradient descent ultimately performed slightly better, which aligns with the SVD algorithms performing the best in the initial algorithm comparison.

Hyperparamter	Optimal Method
method	sgd

Prediction Performance

Using the optimal hyperparameter values, new SVD and Baseline Only models were re-instantiated and trained on the first 70% of the entire dataset. This training data was put through the three initial functions described in this section to create a matrix structure accepted by Surprise algorithms. To allow the training portion to pick up on emerging user trends and build a baseline of ratings, the raw input data was sorted (ascending) by submission date. Specifically, the training set consisted of the first 70% of the ratings and the test set featured the remaining 30%.

The median of rating submissions by users in this dataset is 1. Hypothetically, if the dataset was less skewed in this regard and the typical user had more than a single rating, sorting the data by user and submission date prior to train and test splitting would be recommended. For example, for a user with 6 rating submissions, using the first 5 submissions to train and predict on the 6th recipe would be an appropriate approach.

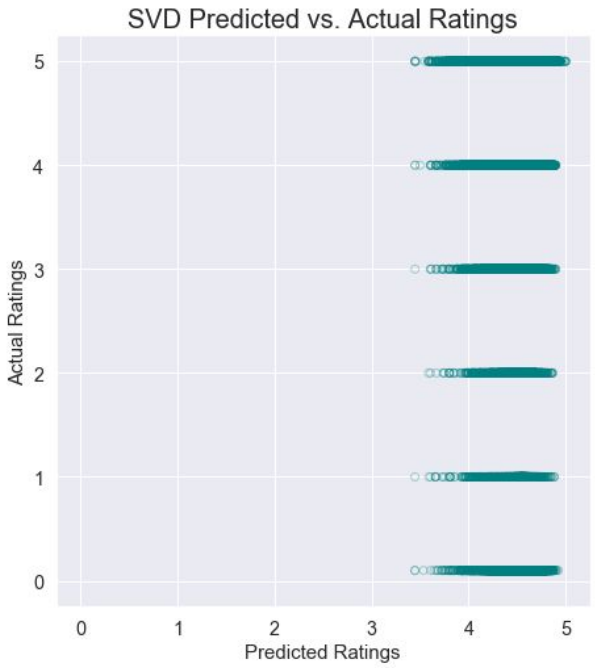
The performance of each algorithm was very similar; however, the SVD model took ~14x as long to train.

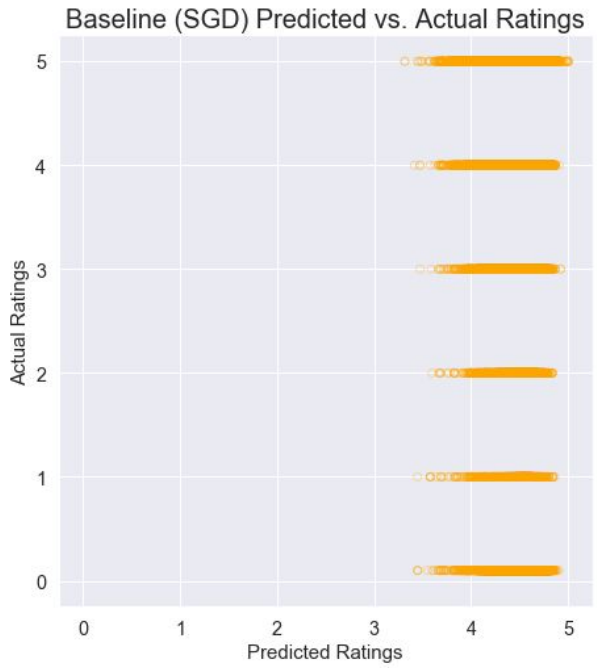
Algorithm	RMSE	Train/Fit Time
SVD	1.0307	50.1 seconds per loop
Baseline Only (sgd)	1.0306	3.5 seconds per loop

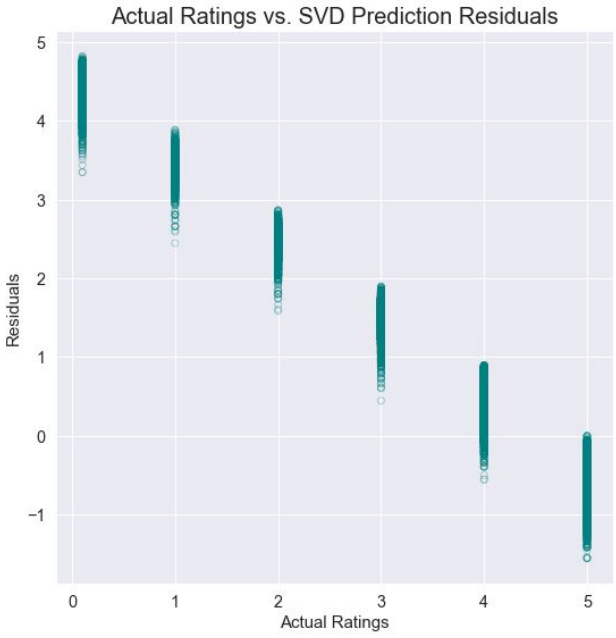
The predictions of the two models were merged for side-by-side evaluation alongside the actual ratings. Below is an initial look at these distributions.

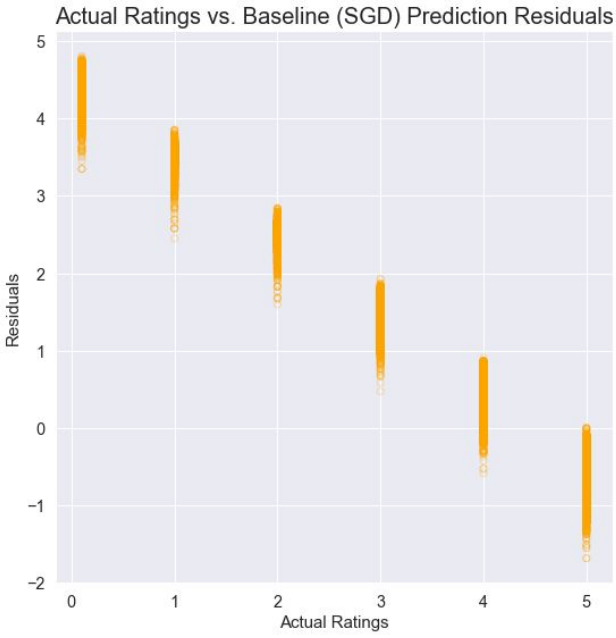
	Actual	SVD Predictions	Baseline Only Predictions
mean	4.534991	4.540851	4.536807
standard deviation	1.033598	0.141208	0.138046
min	0.100000	3.290663	3.315932
25% percentile	4.000000	4.495975	4.491933
50% percentile	5.000000	4.550818	4.546296
75% percentile	5.000000	4.623165	4.616767
max	5.000000	5.000000	5.000000

The most immediate highlight is that predicted ratings do not go below 3 and seem to be grouped between 3.5-5. As mentioned in the exploratory section, the rating scores are strongly skewed towards 4 and 5. This asymmetrical distribution combined with the number of submissions per user skewed towards 1-2 has subsequently skewed predictions to remain high. Thus, re-evaluating with a more balanced dataset of rating scores would be recommended and is done so in a later section.



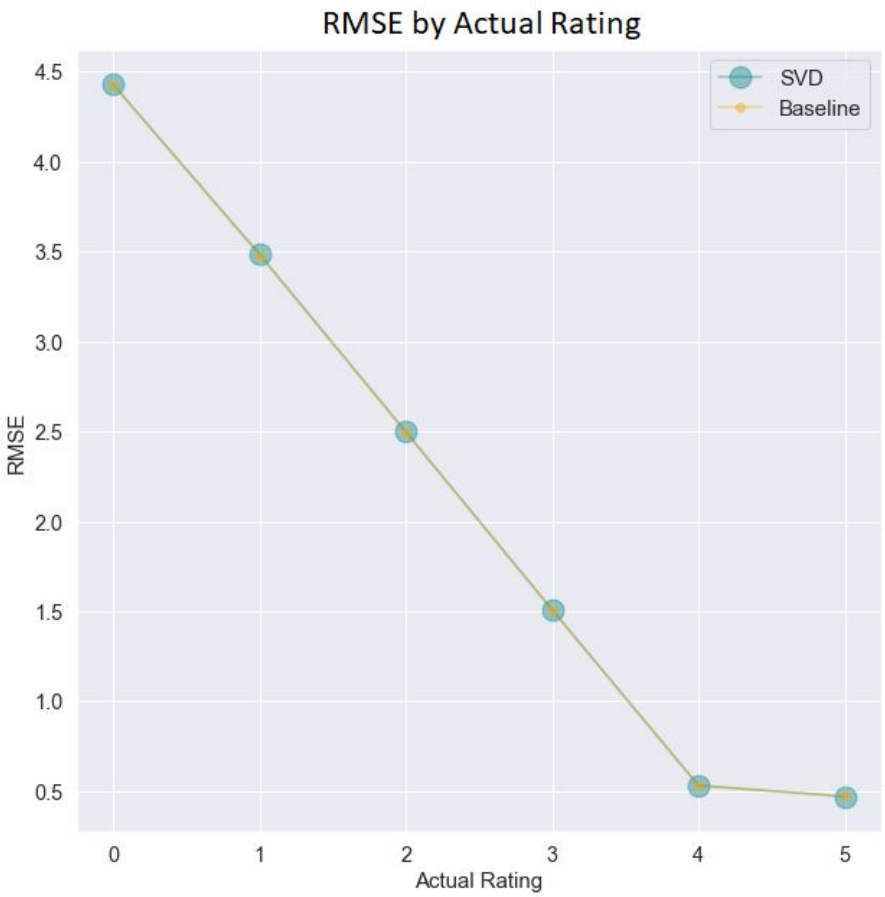






As suspected, the above charts further highlight predictions remaining above 3 regardless of the actual rating. Thus, residuals climb as the actual ratings decrease. The residuals range tends to be around 0 when the actual rating is 4.

Lastly, to confirm that the overall performance of the SVD and Baseline Only algorithm are very similar, the below plots each model's RMSE per actual rating. Thus far, there does not seem to be a significant difference of each algorithm outside of time to train. Later, features are further explored using the Baseline Only model as a result.



Prediction Performance with Balanced Ratings

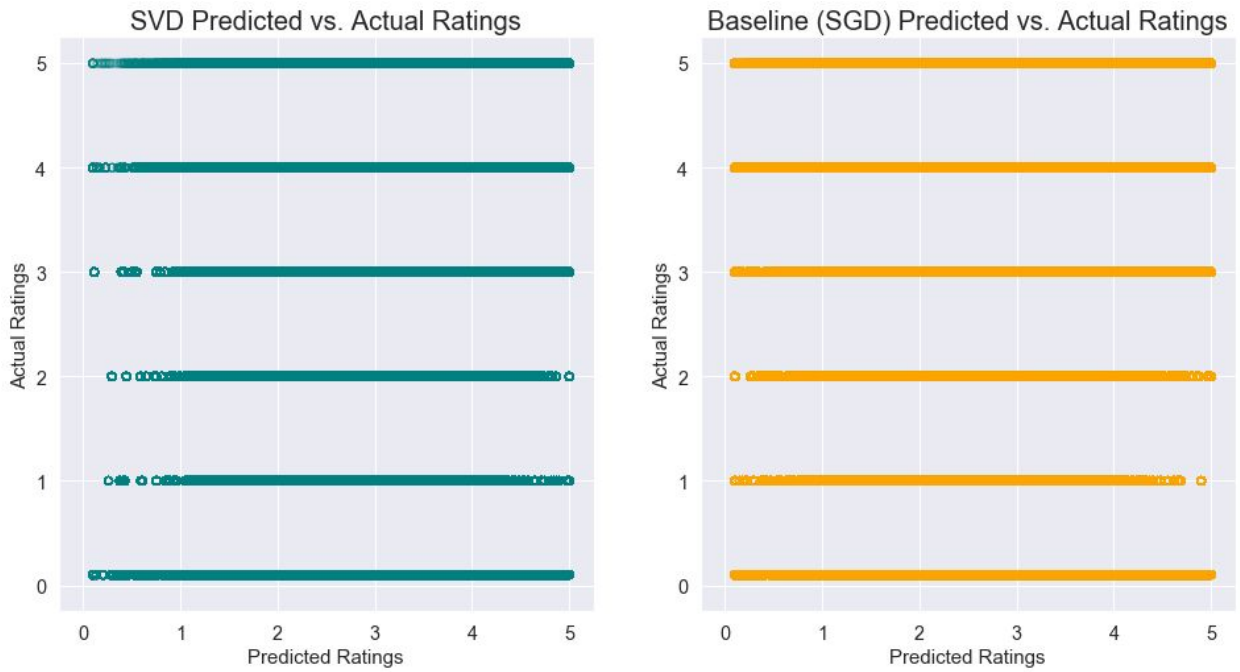
To evaluate the effect of a more balanced dataset of rating scores, equal records of ratings from 0 to 5 were bootstrapped with replacement. This equates to over-sampling of the minority ratings. The same sorting and splitting procedures were run ensuring a 70-30 split of training and testing of ordered ratings. Training time was rather comparable but performance of test predictions was noticeably worse.

Algorithm	RMSE
SVD	1.4832
Baseline Only (sgd)	1.5181

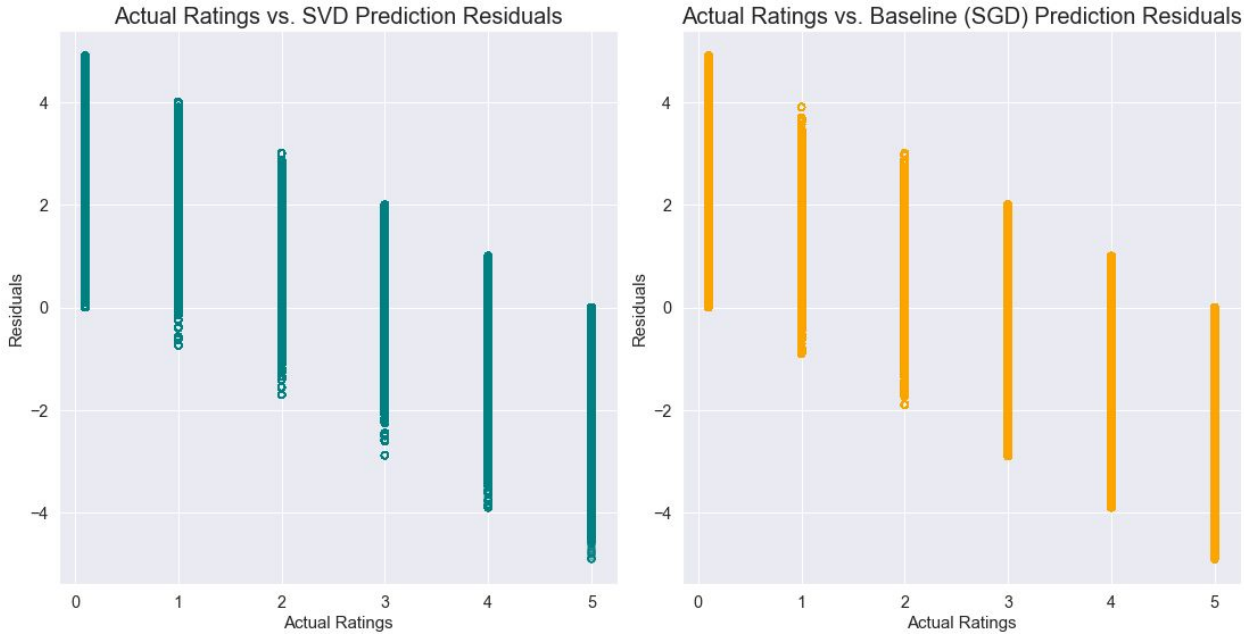
The distribution of the re-sampled dataset predictions are below alongside the actual ratings from the dataset.

	Actual	SVD Predictions	Baseline Only Predictions
mean	1.866434e	2.842635e+00	2.664157
standard deviation	0.9329582	0.7271265	0.7815286e
min	0.1000000	0.1000000	0.1000000
25% percentile	1.000000e	2.274180e	2.274180e
50% percentile	2.000000e	2.740999e	2.550311
75% percentile	2.000000e	3.381486e	3.202690
max	5.000000e	5.000000e	5.000000

Here, the minimum rating submissions drop to mirror that of the bootstrapped dataset. Additionally, the quartile percentile marks all drop. As suspected, another result of balancing the dataset is an increase in prediction variance. This variance tends to increase when ratings become more extreme towards both low and high ratings.



In the original prediction with imbalanced data, residuals close to 0 were congregated around actual ratings of 4. However, using a balanced dataset, actual ratings of 2 through 4 show some portions of their predictions with 0 residuals. Worth noting as well is that recipes with actual ratings of 4-5 were incorrectly rated low more often using the imbalanced dataset. This could be categorized as a falsely negative prediction and could lead to missed recipe opportunity in recommendation generation.



Collaborative Filtering Prediction Takeaways

Both the SVD and Baseline Only models showed significant influence from the distribution of the input data. Specifically, a dataset skewed towards consistently higher ratings results in the vast majority of possible recommendations receiving high ratings, which may lend itself to questioning the value of a collaborative filtering model. Alternatively, a dataset with balanced rating scores results in worse overall performance but a higher variety of predictions.

The above description concerning skewed rating scores coupled with the low rating submissions per user may highlight a key opportunity for further assessment. Specifically, since most users in this dataset are only submitting 1-2 ratings of scores 4-5, the data lack the ability to provide a baseline for recommendation value. Thus, a worthy evaluation would be assessing the performance of collaborative filtering for users who not only submit many ratings but have reasonable distributions of ratings. Alternatively, content filtering may be a preferred route for low usage users while using a popularity filter could be prefered for new users.

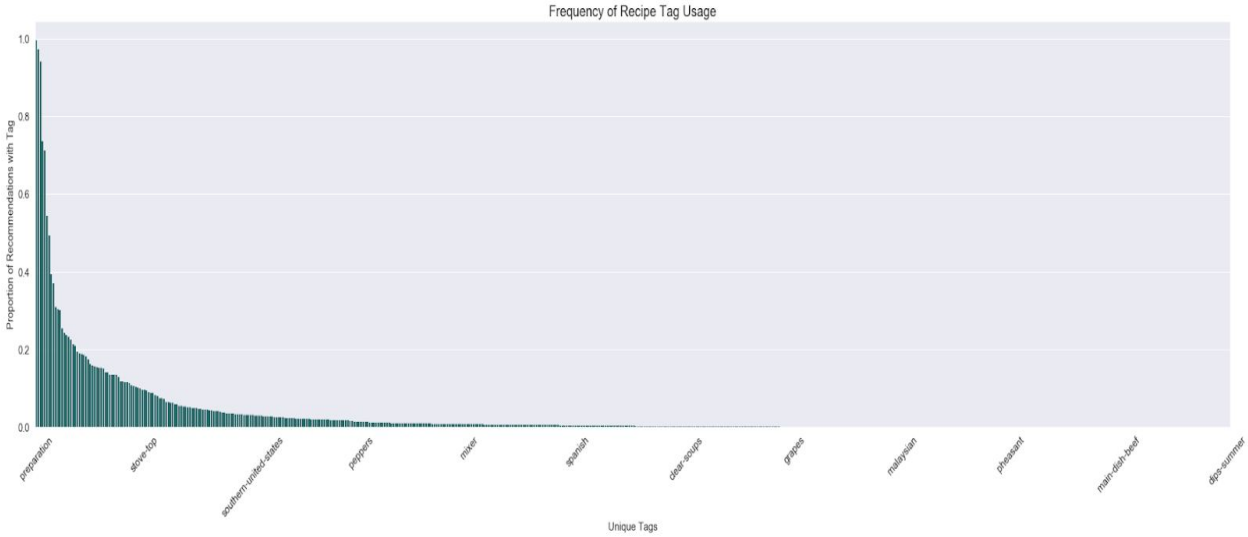
Regardless of the route selected for recommendation generation, a close eye should be placed on evaluating the effects of recommendations. Recommendation feedback, ratings, and customer retention should all be measured and compared to simple recipe aggregation. Additionally, A/B testing should be used for the optimal number of recommendations provided as the differentiation to a user between recommendations and simple aggregations is simplicity.

Content Filtering Modeling

Content Driver Identification

Within the recipe file, each recipe contains information pertaining to its instructions, time, ingredients, techniques, tags, etc. While a large content filtering engine could leverage all of these, the tag field can provide a rather comprehensive differentiator of themes. A single recipe generally will have multiple tags and many tags are shared across recipes.

Below, the chart shows the distribution of tag usage across all recipes. There are 552 unique tags, ranging from consistently used, like ‘main-dish’, or more unique tags such as ‘dips-summer’.



On average, a tag is used in 3.2% of recipes equating to 7,503 recipes. The median is similar while the minimum is a single recipe and the maximum is 99.5% recipe usage. These figures and the above chart highlight that not only is there a long tail of very unique tags but there are widely common tags that should be weighted significantly less in determining similarity.

Similarity Measurement

Due to the above distribution of tag usage, TF-IDF (term frequency–inverse document frequency) is a reasonable approach for assessing similarity of recipe tags because not only does it count usage of tags but it can make the very commonly used tags less important. This should help to counter inflation of similarity due to tags like ‘preparation’ and ‘time-to-cook’. Additionally, a handful of obviously noise words and phrases were captured as a custom stop words list to provide to selected vectorizers. These custom stop words were ['preparation', 'time-to-make', 'course', 'main-ingredient', 'occasion', 'cuisine', 'low-in-something', 'equipment',

'number-of-servings']. As an additional comparison, a simple count vectorizer was used alongside the TF-IDF vectorizer.

To begin measuring similarity, the dataset was parsed down due to resource constraints. Since the similarity procedure has to assess every pairwise comparison of recipes, the data object it creates in process is very large. Thus, the recipes data was parsed down to only commonly rated recipes, i.e. recipes with 30+ reviews.

All spaces and hyphens were removed from the tags field and the tags were transformed to be lower case. With each tags value now simply a list of keywords, it was transformed into a matrix according to the vectorizer. The matrix was then used to create pairwise cosine similarity values of every pair combination of recipe tags. A function was then created to generate the most similar recipe(s) after receiving any given recipe (if contained within the parsed down list mentioned above).

The below chart provides an example of 2 iterations of this function and the corresponding outputs.

Input Recipe	Vectorization Type	Similar Recipe	Shared Tags
boeuf bourguignon a la julia childs	Count	coq au vin chicken braised in red wine	['occasion', 'dinner-party', 'course', 'stews', 'vegetables', 'meat', 'cuisine', 'preparation', 'main-ingredient', 'main-dish', 'french', 'european', 'mushrooms', 'time-to-make']
	TF-IDF		
my thai sesame noodles	Count	easy asian beef noodles	['asian', 'course', '15-minutes-or-less', 'cuisine', 'preparation', 'main-ingredient', 'pasta', 'pasta-rice-and-grains', 'beginner-cook', 'easy', 'time-to-make']
	TF-IDF	thai noodles with spicy peanut sauce	['side-dishes', 'asian', 'course', 'cuisine', 'preparation', 'main-ingredient', 'pasta', 'thai', 'pasta-rice-and-grains', 'easy', 'time-to-make']

Feature Engineering

The below section highlights how collaborative filtering, content filtering, and related processes can be used to provide features to an app or website offering recipe recommendations.

Recommend Top N Recipes through Collaborative Filtering

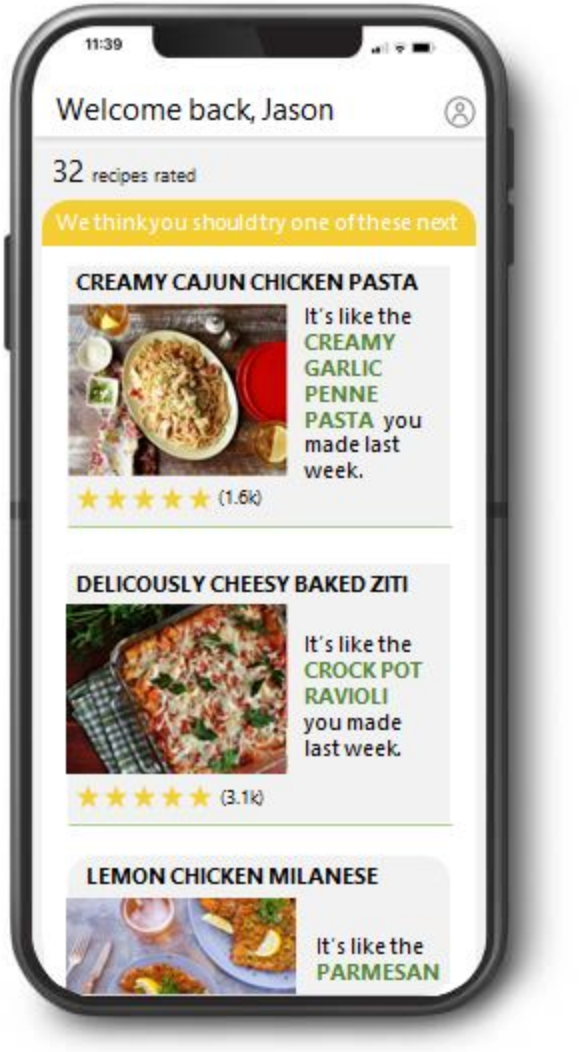
Using trained models that leverage algorithms, such as the SVD or Baseline Only algorithms tuned in the prior section, collaborative filtering can be used to provide a finite number of recipe recommendations for returning users. A function and prerequisite procedure were constructed that take a dataset of recipe ratings, trains a model on the dataset, predicts the ratings for every combination of user and recipe not within the provided dataset, and returns the top n recommendations for each user. Such a function could be the basis of collaborative recommendation generation.

Identify Similar Recipes through Content Filtering

In a prior section, content filtering utilizing the similarity of TF-IDF (and binary count) scores across recipes was explained. A function was subsequently created to leverage these scores to return similar recipe(s). Quite simply, the function will evaluate all pairwise similarity scores for a given recipe in combination with all other recipes (provided to the vectorization procedure) and return the most similar recipe. The function can easily be augmented to return the top n similar recipes.

Merge Collaborative and Content Filtering

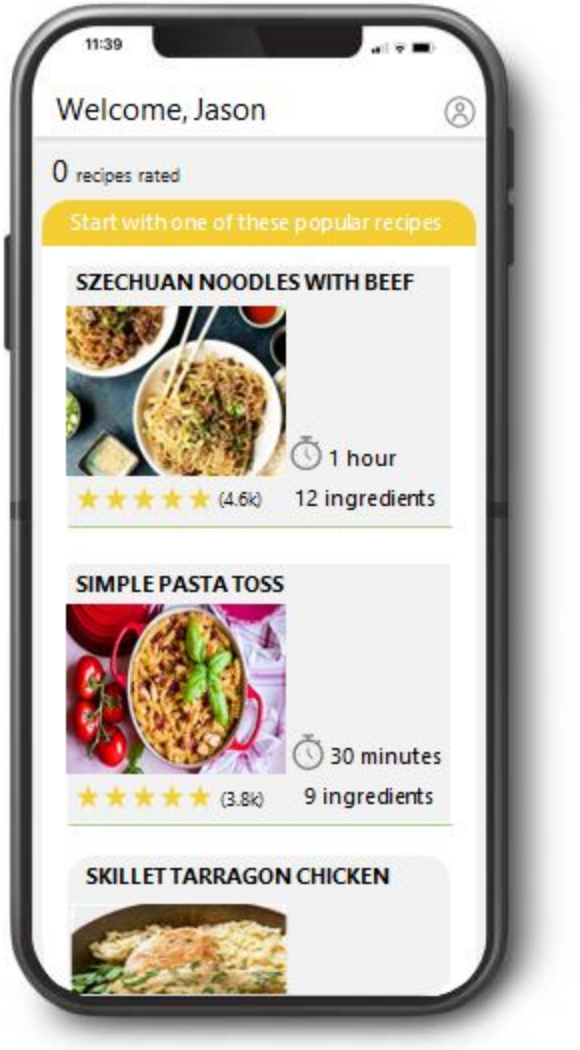
The resulting dataset of the collaborative filtering function above was then combined with the similar recipe generator function above to provide an all encompassing recommendation proof of concept. For a given user(s), the function will return up to 10 recipes that the user has not rated and are predicted to have the highest rating for that user based on collaborative filtering. For each one of these recommended recipes, the function will use content filtering to determine if any of them are similar to a recipe previously rated by the provided user.



Provide Most Popular Recipes for New Users

As noted previously, new (and perhaps low usage) users could benefit from receiving recommendations based on those that are most popular. Additionally, like the raw data used in this analysis, some rating datasets are highly skewed towards high scores. As a result, the mean or the median score of a recipe may be the most appropriate measurement for determining popularity. Lastly, popularity should only be assessed for recipes that have been rated by an appropriate number of users as a baseline.

A function was constructed to receive recipe information, rating information, and user selections of a rating submission threshold, whether to use the mean or median in defining popularity, and the number of recommendations to return. Using both mean and median should again be A/B tested for new users using customer retention and repeat visits to compare effectiveness.



Grocery List Generation

As mentioned in the introduction, there is no single solution that provides a recipe recommendation engine alongside grocery list generation. Thus, a simple function was created that produces a list of unique ingredients necessary based on one or more recipes provided to it. Additionally, if the user already has an existing grocery list, the function will add to it ensuring duplicates are removed. A proper next step for this particular function would address the quantities required for each recipe, ensuring ingredient quantities are added where appropriate.

Conclusion

Simplified recommendations from collaborative filtering and association to a user's history through content filtering could provide a differentiating loyalty driver for large scale recipe

aggregators. However, an organization should fully understand the distributive nature of their recipe ratings and user frequency before pursuing a collaborative filtering engine. Additionally, organizations should implement a different strategy based on popularity filters to capture and retain new customers. The market for digital recipe aggregation is saturated; however, there is room for a highly simplified, highly intelligent recipe suggestion engine.