

School of Electronic Engineering
and Computer Science

Interim Report

Programme of study:

Computer Science and Mathematics

Cooking site that uses a hybrid recommender system with recurrent trend analysis

Supervisor:

Mustafa Bozkurt

Student Name:

Umrhan Sherif

Final Year
Undergraduate Project 2020/21

Date: 01/05/20

Contents

Chapter 1: Introduction.....	
Background.....	
Problem statement.....	
Aim & objectives	
Report structure	
Chapter 2: Related work	
Content-based filtering.....	
Matrix-factorization	
Neural collaborative filtering	
Popularity recommender.....	
Evaluation metrics	
Existing solutions	
Chapter 3: Project requirements	
Functional requirements	
Non-functional requirements.....	
Chapter 4: Implementation.....	
Python & libraries.....	
Django & website architecture	
Website walkthrough with model design	
Considerations & project development.....	
Chapter 5: Evaluation	
Experiments.....	
Final results.....	
Considerations.....	
Chapter 6: Conclusion	
Limitations & future improvements	
Overall conclusion	
References	

1. Introduction

During the COVID-19 pandemic, people remain indoors to keep themselves and others safe. Families are spending more time together, people are more active on social media platforms like Tiktok & Instagram, and people are taking on more recreational activities [27]; one of which is cooking.

1.1 Background

Throughout 2020, there has been a surge of users on platforms such as Tiktok, with 315 million global downloads in Q1 [8]. Tiktok is a social media platform where people share short interactive video content online. Tiktok videos accumulate 1 billion views on average per day, [9] and the content is consumed by users across the globe, with 800 million monthly active users. Users can follow creators, interact with posts via likes and comments, search for posts via hashtags, and save & share videos onto other platforms. Tiktok will curate each user's feed by grouping users into circles based on the category of posts they interact, their watch time on posts with said category, and the creators they follow [10].

1.2 Problem

One circle of users on Tiktok are those with an interest in cooking. Users in these cooking circles will be recommended hot and trending recipes, seasonally relevant recipes, or cooking challenges. Cooking videos on Tiktok are bitesize and quick to digest – but exact ingredients and measurements are not commonly provided. These short videos tend to lack recipe detail; creators often assume you have experience in cooking, and recipes can be tedious to follow. Videos are fast-paced with no rewind functionality; if you miss a step in the recipe, you have to either watch till the end or scroll up & down to watch the video from the beginning which is impractical when you are cooking. Beginner and home cooks may want a thorough recipe guide, whilst still being able to enjoy trending or seasonal recipes.

From my personal experience on Tiktok, trending recipes are a thrill to watch due to the lively presentation and the comedic nature, which leave me eager to try them – but as a beginner cook, I have difficulty following the majority of videos for the aforementioned reasons.

Due to lack of thorough recipe guides on Tiktok, users may proceed to search the web instead. An issue I have found with cooking websites on the web, is that they do not consider the latest trends of recipes found on social media; so those users that are looking for fun and trending recipes to try and share with their friends are left in the dark.

1.3 Aim & objectives

Build a website that can suggest detailed recipes to users based on what is currently popular on Tiktok or Google Trends and recipes they have rated highly in the past. I will use a content-based filtering algorithm to find recipes that are similar to a recipe a user has rated in the past, by building a profile of the user utilizing natural language processing techniques. This will produce suggestions for recipes the user may like – then, I will use either a matrix factorization model or a neural-network model to learn the complex interactions between users and recipes, to predict the rating of recipes from the suggestions. The highest rated suggestions will be the list of personalised recommendations for the user. I will attempt to combat the cold start problem by introducing the 'currently trending' recipes section, to serve as a starting point for users with no history on the site. This website is intended to be a one-stop shop for beginner

and home-cooks who want to follow a thorough recipe in the traditional format, but also want to keep up to date with the latest ever-growing trends from social media.

Objectives

- Create a fully functional website that contains recipes with easy-to-digest formatting, that expresses an intuitive and inviting design, and allows easy browsing for recipes via:
 - Personal suggestions section
 - Trending recipes section
 - Similar recipes section
 - Similar users liked section
 - User pages
 - Pagination
 - Filter & search
- Create or use a tool that either collects: Tiktok hashtag information daily, or the relative interest of recipes on Google Trends
- Create popularity-based recommendation system for suggesting trending recipes via the information collected
- Explore whether matrix factorization (MF) or neural collaborative filtering (NCF) can learn the complex user-item interactions
- Build a hybrid recommendation system that uses both user-item content-based filtering and a collaborative filtering model (MF/NCF), to suggest recipes to users

1.4 Report Structure

In chapter 2, I discuss the literature surrounding recommender systems, the cold start problem, evaluation metrics to assess recommender systems and existing cooking websites. Chapter 3 discusses the scope of the project, which initial requirements have been met and additional requirements that have been added. Chapter 4 discusses Python libraries used, the Django framework, a walkthrough displaying the main features of the website with model implementation, considerations & constraints, and how the project has developed over time. Chapter 5 contains experiments that were conducted to stress test the systems, along with some evaluation and final results. Finally, chapter 6 concludes with improvements that could be made, some limitations and a summary of results.

2. Literature Review

This project fuses both Content-based Filtering (CBF) and Neural Collaborative Filtering (NCF) to recommend new recipes to users. NCF requires no domain knowledge, as opposed to CBF; whereas CBF provides contextual information to guide recommendations. Using these methods in conjunction can remove some of the inherent weaknesses they hold. However, the cold start problem is not addressed by either of these methods; it describes a situation where a recommender cannot effectively produce suggestions for a new user, due to the lack of information available for that user. The Popularity-based recommender's (PR) aim is to solve this issue by taking into consideration the recent trends of social media platforms by either collecting hashtag information from Tiktok or tracking the interest over time for recipes, through Google Trends.

2.1 Content-based Filtering

Content-based recommender systems try to recommend items similar to those a given user has liked in the past [1]. The notion of similarity between a user and a list of items can be described by a similarity rating, which can be computed once the user is represented with the same set of features as an item. These features are defined by the content of the items, which in the context of this project, can be: recipe ingredients, steps, descriptions, tags and nutritional information.

To represent a user with item features, a user profile can be constructed, which is a collection of items that the user has rated; this has a size of P for all users that have rated at least P items.

To find these features, each item is vectorized using the TF-IDF (*term frequency-inverse document frequency*) measure from Natural Language Processing (NLP). TF-IDF is a statistical measure used to evaluate how important a word is to a document (item) in a collection or corpus [2]. This is calculated by multiplying the frequency of a word in a document and the inverse document frequency across all documents in a collection [13]. This measure has the effect of scoring words low if they are present amongst many documents, and words that appear less frequently are seen as relevant to a particular document.

A user profile is created by: taking an average score for the features in items they have rated & multiplying it with the rating they gave – where each feature initially has a TF-IDF weight. Now that the user and item is represented by the same features, a similarity rating can be computed between the user profile and a list of items – those items that are most similar will be selected to generate a list of recommendations for the user.

2.2 Neural Collaborative Filtering

The goal of collaborative filtering (CF) is to predict the preferences of a user based on the preferences of a group of similar users [3]. The central idea is that a user will prefer items that like-minded users prefer, or even items that dissimilar users do not prefer – assuming preference correlations exist between users.

CF can be broken down into two classes, memory-based and model-based. Memory-based algorithms maintain a database of a user-item interaction matrix, and, for each prediction, perform some computation across the entire matrix. On the other hand, model-based CF uses Matrix Factorisation (MF) to reduce a user-item interaction matrix

into vectors of latent features for both the user and item. The level of interaction between a user & item can be estimated by taking the dot product of their latent vectors.

The interactions used to make predictions are either implicit (clicks, view time, positive sentiment) or explicit (rating or score). This project is concerned with explicit interactions using user-based CF. A given user u will have a set of similar users S , that is determined by the rating history of u . User-based CF takes a subset X of S that has rated an item i – i is recommended to the user u if the rating prediction computed from X is above a threshold rating.

A problem that arises with traditional MF, is that its performance can be hindered by the choice of the interaction function – the dot product [4]. In general, a MF model's performance on predictions can be improved by the addition of bias weights. A user may typically rate items very low, or, an item may be universally popular. Biases can lead to overgeneralized recommendations, and so, weights are attached to take this into account. However, the addition of bias weights to improve performance suggests that the relationship between the user and item interactions may be more complex. A dot product may not be adequate in capturing this complex structure as it combines the multiplication of latent features linearly. NCF attempts to solve this by replacing the inner product with a deep neural network architecture that can learn an arbitrary function from data. The NCF framework utilizes a multi-layer perceptron (MLP) to learn the user–item interaction function and combines this with a generalized MF model. This unifies the strengths of linearity of MF and non-linearity of MLP for modelling the user–item latent structures.

This project aims to join CBF and NCF approaches to attempt to provide a 'strong' recommendation – by using the contextual information provided by CBF and the user–item latent structures modelled by NCF. This, however, does not solve the cold start problem. Where, a new user cannot be recommended an item as there is no history to gauge the user's personal preferences.

2.3 Popularity recommender

To solve the cold start problem, the popularity recommender (PR) algorithm analyses the interest of cooking terms on Google Trends to suggest trending recipes to users – this helps provide a basis that the user's profile can be built around, whilst keeping up with social media trends and seasonal events.

Google Trends can produce seasonal data on the level interest for search terms. One way to evaluate seasonal data to determine if a term is trending, is to use the Seasonal Kendall Test (S-K test) – which is a variation of the Mann Kendall Trend Test (M-K test), applied to seasonal time-series data.

The M-K test is a non-parametric test used to analyse time series data to discover whether the value of interest is trending, by checking if it is monotonically increasing or decreasing over a given time period [14]. Before conducting the M-K test, the data must not be seasonal; that is, there must not be any periodic fluctuations, whether that be hourly, daily or monthly. However, the data collected from Google Trends is recorded periodically (one data point per hour, within a given timeframe); and so, the S-K test is more suitable.

The S-K test will run an M-K test on each season in the data, which in our case, represents each day in the time frame. The S-K statistic is calculated by summing each season's Mann Kendall's score, which has the effect of comparing each day's seasons. For example, 7pm on each day would be compared, as with 8pm, 9pm, and so on.

2.4 Evaluation metrics

Precision and Recall

One way to evaluate a recommender system is by computing the Precision and Recall, which are metrics commonly used to evaluate machine learning models. In the context of recommender systems, precision is the proportion of recommendations that are relevant to a user, which is found by: $\frac{\text{No. recommended items relevant to the user}}{\text{No. recommended items}}$ [15]. Whereas recall is the proportion of relevant recommendations that happen to be in the recommendation set. It is found by: $\frac{\text{No. recommended items relevant to the user}}{\text{Total no. relevant items}}$.

Both of these metrics are dependent on how relevance is defined for a recommendation [16]. The definition of relevance in recommender systems is based on the likelihood that the user has a preference for the item, which makes it a subjective measure – only the user knows their preference. One way of measuring relevance is by setting a threshold where, if a predicted rating for an item exceeds the threshold, it is labelled as relevant to the user, otherwise, irrelevant.

A caveat to this approach is that it does not consider users that are either hypercritical or have different tastes. One way to consider the latter, is by adapting the way the threshold value is defined on a per user basis – this can be done by setting the threshold value just below the score of the top percentile of item ratings given by the user. In chapter 5, I explain my approach to this problem, and define my own way of measuring the relevance for a recipe, based on its contents.

Precision or Recall may be more suitable depending on the system it is applied to. In Information Retrieval, Precision is suitable if the task is to find the set of all relevant items within a fixed set. Whereas, Recall is important if the task is to find all possible relevant items for a user [17]. Recall becomes impractical as the number of ratings scale [16] as the probability that an item is relevant to the user increases the more the user rates items. Therefore, precision-based metrics are more appropriate for K-set recommendation retrieval.

Mean Average Precision At K (MAP@K)

Rather than computing the precision for a single recommendation, the Average Precision At K (AP@K) can be used when provided a set of recommendations of size K. AP@K computes an average over the precision at K-1, K-2, ... K-K+1. This has the effect of measuring the proportion of “correct” recommendations and rewards the recommendation set if it has the most likely relevant items recommended first (at the top of the list) [18]. This metric can be computed for every user in a validation set, and a mean can be computed over this to produce a single metric that measures the recommender’s overall performance – this is called the Mean Average Precision At K.

Normalized Discounted Cumulative Gain (NDCG)

Another metric can be used alongside MAP@K to measure the recommender’s performance – as Precision only measures binary relevance; that is, an item is either relevant or not. There is no nuance as to the level of relevance for a recommendation. NDCG, a metric used in Information Retrieval, can be used to measure the quality of a recommendation set, as it assesses the ability for the recommender to retrieve highly relevant documents (items) to the user. It works on the assumption that highly relevant documents are more useful to the user, when compared to moderately or irrelevant documents [19].

Given a set of recommendations for a user, each item will have a level of relevance assigned based on a subjective measure; this can be various rating thresholds, which are assigned a higher level the greater the threshold is. In chapter 5, I discuss the measure I came up with. Assuming we have a relevance measure, the Cumulative Gain (CG) for a recommendation set is found by summing the relevance scores of all items in the set [19]. For example, given a set with scores: $\mathbf{x} = [3, 1, 3]$, the $\mathbf{CGx} = 3+1+3=7$.

An issue that arises is, a set that has the same scores with a different ordering, will have the same CG value. There would be no way of differentiating between the two sets with this metric, even if one set has a better ordering. Discounted Cumulative Gain (DCG) solves this by penalizing highly relevant documents that are lower in the ordering of the recommendation set [20]. It does this by reducing the relevance of a recommendation logarithmically, the further down the set it is. For example, given the set: $\mathbf{y} = [1, 3, 2]$, the

$$\mathbf{DCGy} = \frac{2^1-1}{\log_2(1+1)} + \frac{2^3-1}{\log_2(2+1)} + \frac{2^2-1}{\log_2(3+1)} = 6.92 \text{ (2dp)}.$$

The problem with DCG, is that the score is not comparable for sets with varying lengths. Therefore, we would need to normalize the score at each position. This is where NDCG comes, using the formula: $\mathbf{NDCG} = \frac{\mathbf{DCG}}{\mathbf{IDCG}}$. The Ideal Discounted Cumulative Gain (IDCG) is best case scenario for the sorting of scores in the recommendation set. For example, given a set $\mathbf{z} = [2, 1, 3, 4, 2]$, the ideal ordering for the set is $\mathbf{iz} = [4, 3, 2, 2, 1]$.

Therefore, to find the NDCG of \mathbf{z} :

$$\mathbf{DCGz} = \frac{2^2-1}{\log_2(1+1)} + \frac{2^1-1}{\log_2(2+1)} + \frac{2^3-1}{\log_2(3+1)} + \frac{2^4-1}{\log_2(4+1)} + \frac{2^2-1}{\log_2(5+1)} = 14.75$$

$$\mathbf{IDCGz} = \mathbf{DCGiz} = \frac{2^4-1}{\log_2(1+1)} + \frac{2^3-1}{\log_2(2+1)} + \frac{2^2-1}{\log_2(3+1)} + \frac{2^2-1}{\log_2(4+1)} + \frac{2^1-1}{\log_2(5+1)} = 22.6$$

Hence, $\mathbf{NDCGz} = 14.75/22.6 = 0.65 \text{ (2dp)}$.

This metric gives us a score between 0 and 1, where 1 is a perfect score. It allows for the comparison of recommendation sets with different lengths and, it assesses the ability for the recommender to suggest highly relevant items to the user at the top of the list. A mean NDCG score can be computed over a collection of recommendation sets, to measure the overall performance of the recommender.

Root Mean Squared Error loss

Root Mean Squared Error (RMSE) is commonly used to evaluate Machine Learning models during training and testing. It is the square root of MSE, which is computed by taking the average of the square error. MSE's penalty is proportional to the square of the error, which gives it a large penalty to large outliers and smoothens the gradients for smaller errors [26]. RMSE gives a less extreme loss for very large values, as MSE loss can over-inflate this.

2.5 Existing Solutions

Yummly

A website whose goal is to help home cooks “*discover and demystify dishes that pique their culinary curiosities*” [21]; it is one of the most popular cooking websites on the web with a diverse discovery engine.

Upon entering, you are greeted with a page that asks for ingredients that you would like to exclude in your suggestions. If you choose to proceed by selecting ingredient(s) to exclude, you will be asked about your favourite cuisines such as Mexican, Italian, or Mediterranean. Continuing, you are asked about food allergies, diets that you follow, your cooking skill, and your cooking goals.

Surveying the user in this way is an example of a Knowledge-based recommender system, where a new user submits information upon entering to build up a profile which will inform future recommendations. This can be done by: assigning tags to answers that the user gives, keeping account of preferences specified, and filtering items the user is certain not to try or like (due to allergies or filtered ingredients).

Yummly also includes popular recipes on the home page, which is similar to the PR, minus the external trending information.

Cooking New York Times

A cooking subscription service that provides premium recipes from top level chefs and New York Times (NYT) editors, both on the web and mobile.

Entering as a guest user, you are greeted with: a handwritten recommendation for what you should try through the week, a section from their team that displays recipe collections, and cooking guides for novice or advanced cooks. The recipe collection section has seasonal collections such as Mother's Day recipes and Ramadan dishes. Other collections are their 10 most popular recipes to date and collections for specific categories such as vegetarian cuisines, kid-friendly meals, and healthy breakfasts.

All recipes and collections have a 'save to recipe box' button, which is a feature available for signed in users to keep a bookmark of recipes for later use. CNYT uses these saved recipes to build a user profile for the user to serve personalized recommendations to users on the home page under the 'Recipes we think you'll love' section.

Recipe pages have a rating and review section where comments are sorted by their 'most helpful' rating, and a signed in user can rate the recipe. The page also has sections for 'recipes to try' based on the collection the recipe belongs to, and a 'trending recipes' section for recipes in a certain category such as cooking or baking.

Kitchen Stories

All recipes are 'Stories', which are high-production video content pieces made by editors at Kitchen Stories. Each Story contains a step-by-step recipe video, traditional recipe details, how-to videos, and a 'More delicious ideas for you' section.

The how-to videos are tips and tutorials to help prepare the dish. They are selected for the Story if they are relevant; this could be either selected manually or using a sort of content-based algorithm, where how-to videos are selected based on keywords that match the Story's description, recipe steps or ingredients. As an example, 'How to cut peaches', 'How to secure a cutting board' and 'How to preserve fresh herbs' are how-to videos for the recipe 'Rosé spritzer with white peaches and basil'.

'More delicious ideas for you' section contains similar Stories. This could be done using a content-based algorithm that considers recipe tags and author, as the Stories shown are of the same category of foods/drinks and a majority is made by the same editor. For example, for the 'Rosé spritzer with white peaches and basil' story, the similar Stories shown were 'Rhubarb soda', 'Peach iced tea', 'Tequila sunrise' and 'Mexican beer cocktail'. All of which, are fancy drinks made by the same editor.

Summary

	Pros	Cons
Yummly	<ul style="list-style-type: none"> • Knowledge-based recommender • Content-based recommender • Trending recipes from their site 	<ul style="list-style-type: none"> • Home page is unorganized with scattered recommendations
Cooking NY Times	<ul style="list-style-type: none"> • Content-based recommender • Variety of recipe collections 	<ul style="list-style-type: none"> • Signup required for recommendations • Paid service
Kitchen Stories	<ul style="list-style-type: none"> • Recipe content in the form of original social media Stories • How-to videos • Similar recipes are categorized intuitively • Community-centric 	<ul style="list-style-type: none"> • No personalized recommendations

Figure 1. Comparing cooking websites side-by-side.

<https://www.yummly.co.uk/>

Functional and non-functional requirements that were defined at the beginning of development, to prevent scope creep and to define constraints for operation.

3.1 Functional Requirements

This table of requirements describes the scope of the project. Partially complete or incomplete requirements will have a reason to explain why.

	Requirement	Complete	Reason
1.	Guests should be able to sign up for an account	Yes	-
2.	Guests must log into an account before they can use the website	Yes	-
3.	The website must send a confirmation email for a user that has signed up	Yes	-
4.	Users should be able to log in and log out of their account	Yes	-
5.	Users should be able to search for a recipe	Yes	-
6.	Users should be able to filter a search based on minutes to prep, date of submission, common ingredient and nutritional information by DPV (calories, saturated fat, total fat, sugar, carbohydrates, protein, sodium)	Partially	The user can filter their search with all categories except nutritional information. It was not at the top of my priority list and would require me to make a lot of changes to the database models. Given more time, I would implement this.
7.	Users should be able to view the full contents of a recipe	Yes	-
8.	Users should be able to share a recipe to their social media account (Facebook, Twitter, Instagram)	Yes	-
9.	Users should be able to rate a recipe	Yes	-
10.	Users should be able to comment on a recipe	Yes	-
11.	Users should be able to view their rating history on their account settings	Yes	-

12.	The website should record the history of ratings for a user	Yes	-
13.	The website should show a 'Trending Right Now' section to all users	Yes	-
14.	The website should not show a 'Recommended For You' section if the user has not rated a recipe on their account	Yes	-
15.	Users should be able to view and rate a 'Trending Right Now' or 'Recommended For You' recipe	Yes	-

Figure 2. Table of functional requirements.

3.2 Non-functional requirements

Table of requirements that define the criteria used to judge the operation of the system.

	Requirement	Complete	Reason
1.	Function calls on a separate thread must take no more than 5 minutes to complete	Yes	-
2.	Each recipe page should load in no less than 200ms	No	I did not consider the time it could take to fetch data. When a recipe page is loaded, the website will: fetch recipe details, web scrape food.com for additional recipe details, load in similar recipes, load in similar user suggestions, compute average ratings for all visible recipes, and display user comments. The actual time is ~500ms.
3.	Website should be easy to use, with minimal buttons	Yes	-
4.	Website must be operable on any web browser	Somewhat	Due to the way the web template files that I borrowed were written, the website is most suitable for Microsoft Edge. It is operable on other browsers, but the html elements are not positioned as intended.

5.	Website should not collect personally identifiable information other than the user's username and email address	Yes	-
6.	User confirmation email must be sent within 10 seconds	Yes	-
7.	Website must authorise users using the log in system	Yes	-
8.	The website will be written in python, to ensure readability and improve maintenance	Yes	-

Figure 3. Table of non-functional requirements.

Additional requirements

The website meets all of the important functional requirements that I set out in my project's initial scope, but since then, I have made a few additions to the website that fit in the project's scope:

Added a 'See all recommendations' page, accessible from the home page, to allow the user to browse more than the four recommendations they would see on the home page. A comment section has been added under each recipe, which shows user reviews for that recipe, to help the user decide whether they want to try the recipe. Users can delete their own reviews, which will have a cascade effect, to ensure the average rating and comments are updated for the recipe which the review was associated with. Lastly, I allowed a user to view any user's profile page in case they want to try recipes reviewed by another user.

4. Implementation

A section detailing my methodology for implementing the objectives set.

4.1 Python & libraries

Python is a high-level versatile programming language with a wide range of support libraries, suited for multiple application purposes. It was used in this project for data mining (data cleaning, reduction and transformation), web mining, machine learning, neural networks and building a website backend using Django. Other libraries used in this project include: Pandas, Matplotlib, Scikit-learn, Numpy, Scipy, Fastai, Librecommender, Pymannkendall, Pytrends, Joblib, BeautifulSoup, Selenium, Requests, and Python's default libraries (time, multithreading, json).

Pandas, Numpy, Scikit-learn, Scipy and Matplotlib allowed me to manipulate objects that contain data to build the algorithms for: recommender systems, working with user and recipe data from a database, functions that produce evaluation metrics and plotting results. Librecommender and Fastai were libraries used for matrix factorization and neural collaborative filtering models. Pytrends was used to retrieve data reports from Google Trends to be used with the S-K test provided by the Pymannkendall library. Joblib was for saving and loading python objects, which was needed by recommenders and website backend scripts during use. BeautifulSoup and Requests was used for web scraping food.com to collect images for all recipes used in the database and Selenium was used for dynamic web scraping of Reddit (for testing) and Tiktok (for cooking related hashtags). Lastly, time was used for script benchmarking; json was used for transferring datasets from Google Colab that I had performed data cleaning, reduction and transformation on; multithreading was used to execute recommender scripts when users created, updated or deleted reviews for a recipe on the website.

4.2 Django & website architecture

Django is an open-source high-level Python Web framework used for rapidly developing a secure and scalable website, with robust design [22]. Django's project structure follows the Model-View-Template (MVT) architectural pattern where, the Model represents a database, the View handles user interactions and models to satisfy user requests, and the Template generates HTML using the provided HTML and the Django Template Language (DTL) [23]. Templates use the DTL to perform operations on data that is provided by the View, using python-esque syntax and template tags; these are python functions that can be passed data, and can return python objects to be used or iterated upon, to generate HTML that is subjected to programming logic.

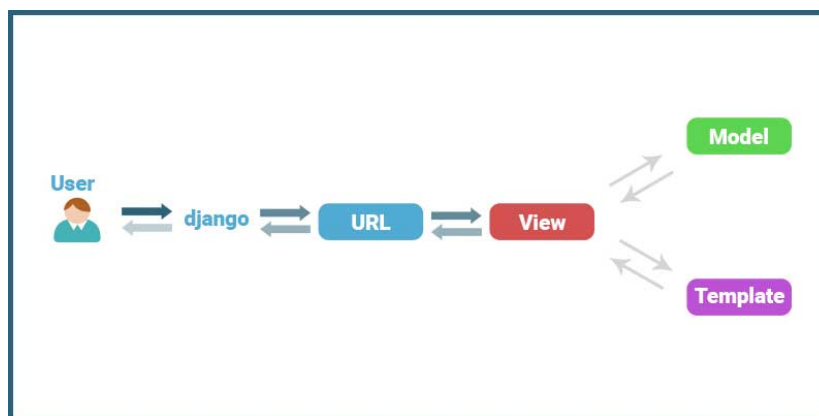


Figure 4. Illustrates how each of the components of the MVT pattern interacts with each other to serve a user request.

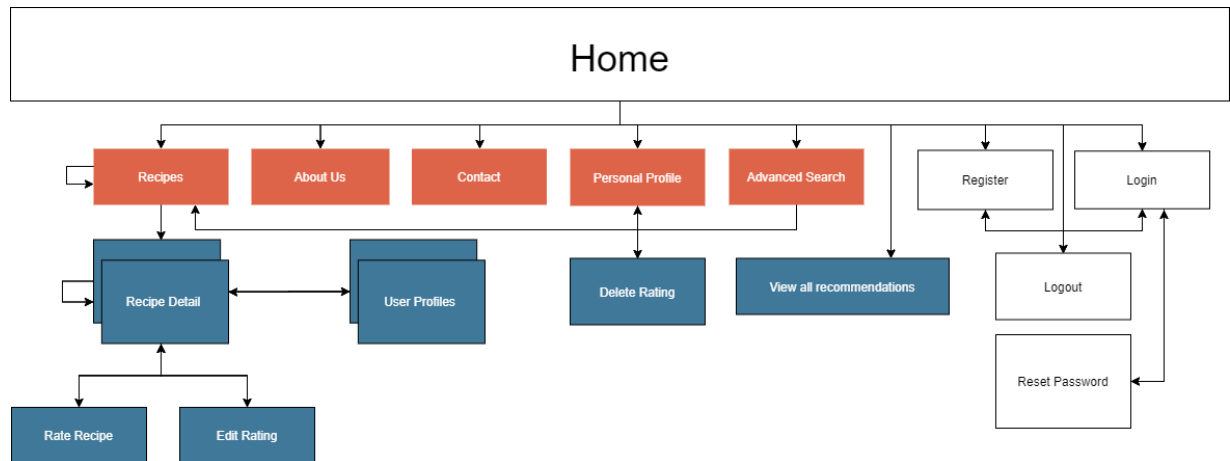


Figure 5. Sitemap of the website.

Figure 2 shows all Templates associated with the website and each link represents the redirects that are made by the View, when clicking a link or button. Self-links represent pages that are either paginated (Recipes) or redirect to other pages of the same kind (Recipe Detail).

The View contains the python logic that maps a URL to a Template and provides context data to be used by the DTL. As an example, for the home page, the View will command the Django server to load in the currently trending recipes and the user's profile recommendations (empty object if there is no recommendations data associated with the user), via the Joblib library and the Model database. This data is stored in a 'context data' dictionary and is passed to the Template that the home View function is mapped to. The Template will call template tags which contain python functions to: load images, format data to a particular string format, return lengths for if-else statements, perform arithmetic operations and to provide logic needed to generate the correct number of html elements to show the correct star ratings for a trending recipe. Once the data from these template tags are used and iterated upon, HTML code is generated to be rendered by a web browser.

4.3 Website walkthrough with model design

Home page

The home page displays trending recipes and user recommendations, provided they have rated at least three recipes in the past; this is needed to build a basic user profile. These user recommendations are produced by using a hybrid of the CBF and NCF model. When a user rates a recipe, the `UpdateUserProfileRecommendations.py` script is triggered which will sample five of the most recent recipes the user has rated, those of which, are recipes where the rating given by the user was above their mean rating score - which suggests a mild preference for the recipe.

This sample of recipes is used to build the user profile, by extracting features from each recipe's ingredients, steps and description. Features are weighted using the TF-IDF measure and a multiplier which is scaled based on the preference level.

For example, if a user rates recipes (out of 5) with scores: [2, 4, 4, 2, 5], the mean rating is $\frac{2 + 4 + 4 + 2 + 5}{5} = 3.4$. Recommended recipes with a predicted rating above 3.4 will be selected for the user profile. The features in recipes will be weighted by the TF-IDF measure and a multiplier, which is a floating-point value been 1 and 3, scaled by the recommendation's predicted rating. Eg. If a recommendation has a predicted score of 4.6, the multiplier would be calculated as $\frac{4.6 - 3.4}{5 - 3.4} + 2 = 2.75$.

Once this user profile is built, it is treated as a recipe, and so it is added to the recipe dataset. A cosine similarity matrix is computed with all recipes in the dataset, and fifty of the most similar recipes to the user profile are selected. Before performing predictions on the ratings for these recipes, the user's rating history is fed into the model to retrain it, so that similar user recommendations can be made. Upon completion, a predicted rating score is computed for each recipe, based on what similar users enjoyed. The top six highest rated predictions, that are most similar to the user profile is then produced, to be viewed on the home page.

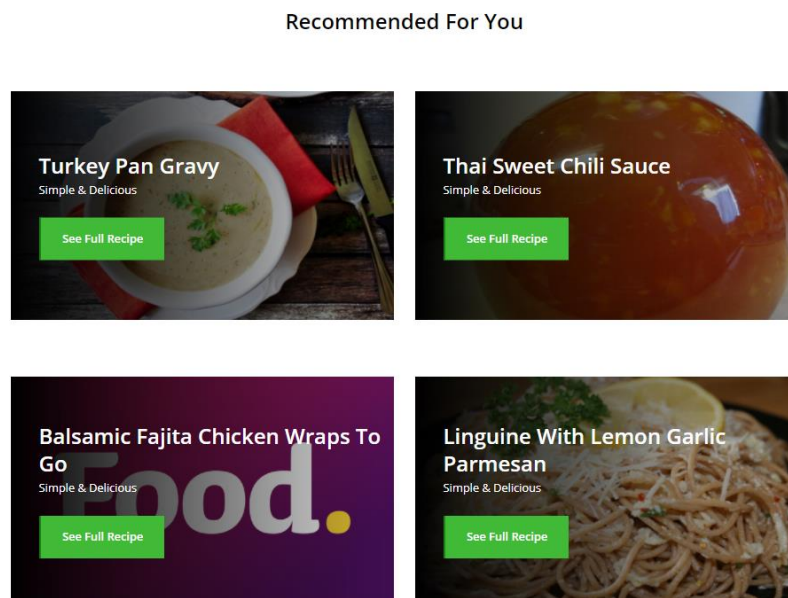


Figure 6. Example of recommendations that are shown to a user on the home page.

Six trending recipes are shown to the user based on whether the recipes contain any of the two top trending ingredients/keywords (terms) from the past day, and how high their popularity score is on the website. The top two trending terms are determined by feeding a set of recipe terms into an algorithm that will: extract a three-day report of the level of interest for each term (through Google Trends, using the python library Pytrends), compute the Seasonal Mann-K score for each term using their report, and sort terms in descending order of Mann Kendall S scores.

Once the top two trending terms are found, the popularity score is found by computing the average rating score and multiplying by the number of ratings. Recipes that contain any of the trending terms are finally sorted by their popularity scores in descending order.

Trending Recipes



Pasta With Sausage Tomatoes And Cream
★★★★★ (426)



Olive Garden Pasta E Fagioli Soup In A Crock Pot Copycat
★★★★☆ (300)



Easy Pizza Pasta Casserole Oamc
★★★★★ (147)



Cheese Tortellini Pesto Pasta Salad
★★★★☆ (147)



Kittencal S Italian Tomato Pasta Sauce And Parmesan Meatballs
★★★★★ (73)

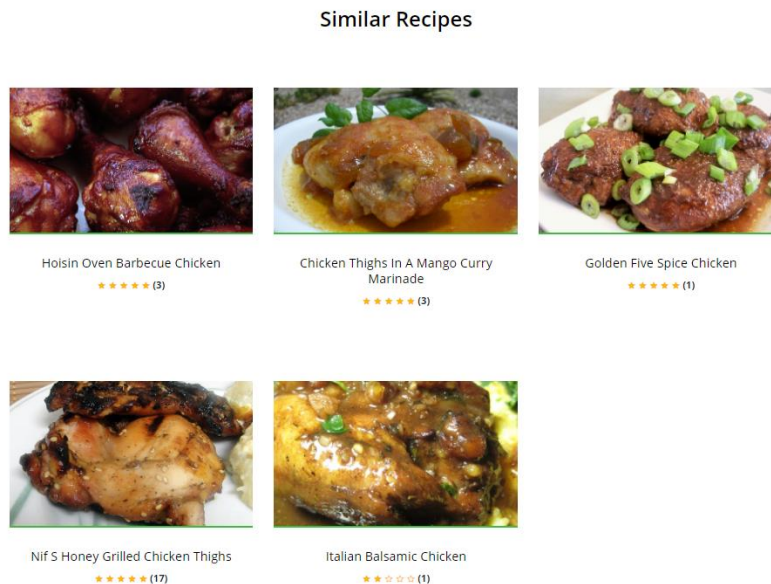


Lemon Shrimp Pasta
★★★★★ (40)

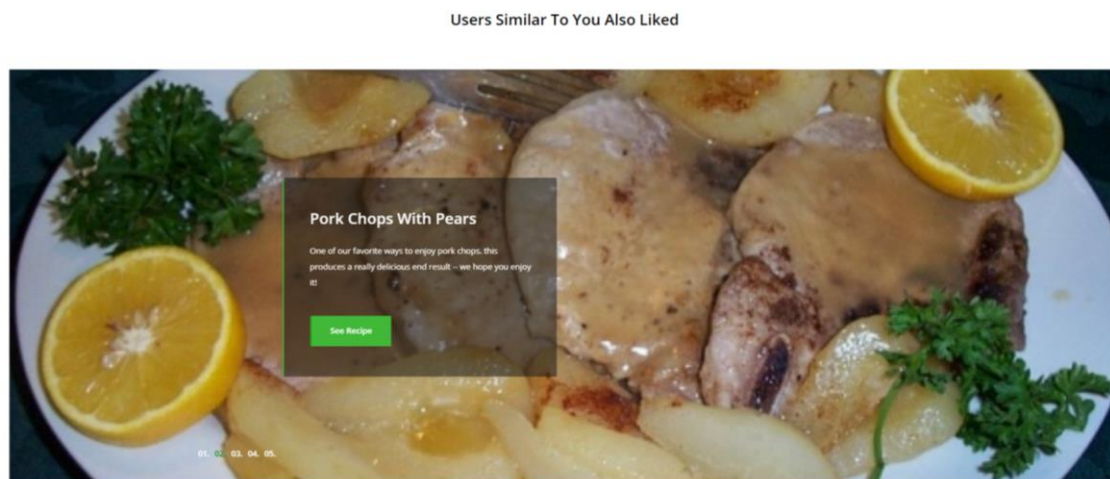
Figure 7. Trending recipes that are shown to a user on the home page.

Recipe detail page

On the recipe detail pages, as well as including the recipe info and comments, the user will see two recipe suggestion areas: most similar recipes to the recipe being viewed and recipes that similar users enjoyed using the NCF model (independent of the CBF, unlike the home page recommendations).



The similar recipe section shows up to six recipes using purely a cosine similarity matrix. These recipes are pre-populated under a 'related recipe' field beforehand, rather than loading the similarity matrix in memory, to improve performance.



'Similar users also enjoyed' section is updated whenever the user submits a recipe review. But, unlike the hybrid recommender, I decided that the rating does not need to be above the user's mean score, as the recipes the user does not like will affect how the NCF model will model similar users – which, may decrease the likelihood that a recipe the user will not want to try will be shown.

The assumption that I've made here is that: the user will not want to try recipes similar to those they have disliked in the past.

Personal profile page

Profile page lets the user update basic details, update trending recipes (only on admin account) view all ratings made, redirect to a recipe that was reviewed, and delete reviews. The update trending recipes button will trigger the `UpdateTrendingRecipes.py` script for demonstration purposes. In real world deployment, the script would run once a day, without the need for an admin to press the button.

When a review is deleted, the `UpdatedUserProfileRecipes.py` (only if the rating was above the user's mean) and `UpdateSimilarUserRecommendation.py` scripts will be triggered, to ensure that there is no influence in recommendations from the recipe that the review was removed from.

Your Reviews







		
Easy Asian Skillet Chicken - 153616 ★ ★ ★ ★ ★ DELETE REVIEW	Fiesta Baked Chicken - 404034 ★ ★ ★ ★ ★ DELETE REVIEW	Snow Peas And Chicken - 12430 ★ ★ ★ ★ ★ DELETE REVIEW
		
Onion Frittata - 28450 ★ ★ ★ ★ ★ DELETE REVIEW	Baked Prickly Pears - 104089 ★ ★ ★ ★ ★ DELETE REVIEW	Almond Fudge Banana Cake - 142 ★ ★ ★ ★ ★ DELETE REVIEW

Figure 10. Example of the rating history section for a user's profile.

4.4 Considerations & project development

Datasets used

Kaggle provides a dataset that contains 230K+ recipes and 1.1M+ recipe reviews, dating back at most 22 years, scraped from the website “Food.com” [12]. All recipes and user comments on the website are from this dataset, with no personally identifiable information. I pre-processed and randomly sampled 20% of all recipes from the dataset and stored it in a SQLite database to be used by the CBF model and in the website. The user interactions dataset (user reviews) is used for training and testing the NCF model and can be viewed under each recipe on the recipe detail page.

All recipe images on the website have been web scraped from Food.com using the fact that each recipe url starts with the recipe name and the recipe id – this allowed me to scrape all recipe images. A proportion of recipes have no image on the site hence they were given the ‘Food.’ image. A very small percentage of recipes on the site have since been deleted, and so they have the default ‘Food.’ image too.

Space & memory needs

Due to the sheer size of the datasets, I have had to strip down the number of recipes by sampling 20% from the “raw recipes” dataset and have omitted user interactions that are not from these recipes. As well as this, I have split this 20% sample into two equal size sets; one set is used by the CBF model to produce a cosine similarity matrix with a file size of 4.24GB, and both sets are used to train the NCF model.

The file size of 4.24GB is not an issue for my storage requirements, but it is necessary to load this matrix in memory to produce recommendations. When evaluating the models, I am only able to load in at most two at a time; it takes a considerable amount of time to evaluate, and so, I have decided this size is sufficient. Although, the NCF model does use a lot less memory, and so I decided to use all user interactions for the 20% recipes set for training.

These decisions were made for ease of use, but it does come at the cost of performance for the models. With access to more resources, I would use all raw recipes & user interactions.

User-friendly considerations

Before embarking on this project, the user friendliness of the website was of deep importance to me. And so, I have ensured that: I use the minimal number of buttons needed to perform actions on the website, the colour palette is intuitive and consistent for all user interactive elements, and lastly, the font size is large enough with clear separation for recipe steps, ingredients, suggestions and comment sections – to improve ease of use.

Pairwise similarity metrics

Sklearn, a popular ML python package, provides a variety of pairwise similarity metrics that can be used rather than the cosine similarity metric. After exploring several metrics such as Sigmoid Kernel, Linear Kernel and the Polynomial Kernel; I found that the

recommendation sets provided to users would not differ. Results would not vary, and the size usage would remain the same, and so I settled on the Cosine Similarity metric.

Old content-based recommender

With regards to the content-based recommender, I will be computing the cosine similarity between recipes and a given user profile by comparing their features: Ingredients, Recipe Steps, Recipe Description, Recipe Tags and Nutritional Information. This user profile will be a vector that contains the same set of features as recipes, for comparisons to be made. It is found by computing the weighted average of the currently rated recipes by the user. where, each weight represents a rating – but each rating normalized using the user's average rating, to consider user biases.

This algorithm will produce ten similar recipes and will be fed into the NCF model with the user. The NCF model will produce rating predictions based on similar users. The top three recipes with the highest rating predictions, will be the list of recommendations for the user.

FastAI & issues faced

Initially, as I was learning about machine learning and AI, I came across the FastAI programme, which aims to make learning AI approachable and accessible. FastAI is free and open source and claims to simplify the training of fast and accurate neural networks using best practices [11].

FastAI has a class “Learner” which is responsible for the training of all of the neural nets they provide, as well as preparing batch data for evaluation, and saving/loading models. In order to work with tabular data (rows and columns of data), FastAI has a module “Tabular” which aims to provide the tools needed to process tabular data, which is necessary for the data that I have from the Kaggle dataset. Tabular has its own dataloaders which allow data to be split into batches, to be used in a training loop; as well as its own TabularLearner (inherits from Learner) which has its own functions to perform predictions on tabular data.

Piecing this all together, FastAI's “Collab” class inherits from the Tabular module for collaborative filtering – it provides both a dot-product model & a neural net, by tweaking the parameters of the collab_learner function.

Regarding my use of the FastAI python package, I first pre-processed the user interactions data by first removing all features except the user id, the recipe id, and the rating given (target feature). Then I normalized the target feature by mapping ratings from 1 to 5, between 0 and 1. Lastly, I fed the data into a ColabDataLoader for use. I created a MF model using the dot-product variant, and I started tuning hyperparameters such as the number of latent features, the optimizer, the learning rate, the number of epochs and the batch data size. I followed the same procedure with the neural net variant, but instead, I tweaked the size and the number of hidden layers. I repeatedly trained both variants observing both the training loss and validation loss, being mindful that if the training loss exceeded the validation loss it meant that I was overfitting to the data; and that, too high of a training loss meant I was underfitting.

In the end, the validation loss of the MF model was comparable to the loss of the NCF model. Using the tuned hyperparameters, and training 20 times, the average validation loss (RMSE) was 0.288 for the MF model, and 0.285 for the NCF model.

Now I began making predictions on user test data for inference. What I found was, the predictions made were mostly the same. There was rarely any distinction in predicted ratings, and they were skewed towards a 5 score. Another issue I found was, FastAI no longer had support for retraining the model. In order for me to deploy the model on my website, I would need to load the model, retrain it with new user data, and save it – this was no longer possible in the latest version of FastAI.

I attempted to use an earlier version of FastAI where retraining was possible but found that the average validation (RMSE) loss jumped to 1.421 for the MF model and 1.319 for the NCF model, using the same hyperparameters. Predictions on test data were seemingly random; overall after many tries with hyperparameter tuning and testing, I deemed this attempt a failure, and moved on to find an alternative library, which led me to finding LibRecommender.

Current use of LibRecommender

After some digging, I found LibRecommender, which is an easy-to-use python API that contains a range of recommendation algorithms, with low memory usage, and the ability to easily retrain models with new users and items [24]. LibRecommender uses a package called Libreco, which is an open-source python package that houses the algorithms used in LibRecommender.

Using the LibRecommender API, I initialized both a MF model (labelled as SVD in library) and an NCF model. I decided on the NCF model after evaluation: training for 6 epochs, tuning hyperparameters and training 10 times, the average validation RMSE loss was 1.0371 for the MF model and 1.0056 for the NCF model. Before training and evaluating the model, a trainset and testset is built using the DatasetPure class from Libreco. The NCF model is then fit to the trainset, and inference can be made.

Predictions can be made on recipes for an existing user, a new recipe for an existing user, or an existing recipe for a new user. Librecommender's inbuilt `recommend_user` method provides a set of k item recommendations based on the highest predicted ratings for the user. Both the `predict` and `recommend_user` methods have a parameter 'cold_start', which has two options: "average" and "popular". Average will use the average behaviour of all users (of all user/item embeddings) to attempt to solve cold start problem [25]. Popular will just return the most popular items in the trainset. I did not use either of these options, as, I aimed to solve the cold start problem with my popularity recommender, that takes into account trending ingredients as well as the universally popular recipes amongst the trainset.

Issues faced with popularity recommender

Initially, the popularity-based recommender would work by feeding the most recent cooking-related hashtags from TikTok into an algorithm to compute a popularity score. Then suggestions are made to all users/guests on the website, with recipes that contain the hashtag in their ingredients or description, sorted in descending order by their popularity score.

To collect these hashtags, I built a web scraping bot that gathers hashtags from the Tiktok trending page. The bot would command a web driver to scroll for a defined amount of time to load the content of the page, and then render the JavaScript from the page source as HTML, for information retrieval. I incorporated random pauses and velocity shifts to simulate a human scrolling & stopping on the trending page, to avoid bot detection.

Problems arose, as Tiktok recognised that I am using a bot and will not allow me to login, even if I: rotate IP addresses & my User Agent, remove cookies, create a new account, or run the script on a separate device. As I am unable to log in, the hashtags I extract will not be relevant to cooking - because a Tiktok account will have a personalised feed that tracks the content that you engage with via likes & comments, watch time, and who you follow. A guest user will only see popular comedy videos from Tiktok stars and celebrities. The end goal is to extract hashtags from the feed of a user who is interested in cooking, to find out which hashtags appear most frequently, to base suggestions on.

Alternatively, I experimented a multitude of unofficial Tiktok APIs to extract hashtags, but I ran into the following errors: My Tiktok account's session cookies were not valid, the Selenium browser used would give a 'closed unexpectedly' error message, a website's API would block requests under a paywall, error messages such as 'TikTokApi' object has no attribute 'width' or "Runtime error: this event loop is already running" occurring unexpectedly, APIs not working suddenly due to Tiktok changes and lastly my IP address getting blocked with only a few requests as rotating IPs found online was infeasible in the long run.

I deemed this as overall unsuccessful, and instead switched to the backup plan, which was to use Google Trends as a means for determining what recipes to suggest. I did not choose to scrape from Google Trends, as I wanted to avoid Google blocking my IP address during the testing phase and so I decided on Pytrends, with light use to avoid this issue.

5. Evaluation

To measure the effectiveness of the CBF and NCF models, I have evaluated them using the AP@K and NDCG@K metrics with existing users from the website database. I put together seven experiments that test the Hybrid model and the CBR on various user types & parameter values.

5.1 Experiments and analysis

Methodology

These experiments work by conducting tests (plots) that evaluate the Hybrid and CBF model with varying threshold values, to measure how the models fair when stretching these values to extremes; this is especially necessary for these models as they rely on a 'relevance' measure that is subjective due to the idea that there is no concrete way to define how relevant a recommendation is to a user – only the user knows this.

My attempt at defining how relevant a recipe is to a user is by considering the percentage of features that is in common between a recommendation and a recipe from the user's profile. This is not the same as measuring the similarity of a recipe with the user profile, as with that similarity measure, list of possible features would be weighted by a rating score, a TF-IDF weight and the number of recipes that share that feature. Whereas in this case, the weight of each feature is not considered.

Percentage of features that are in common between two recipes is subjective, and so, I have attempted to consider multiple 'threshold' values in each test, to hopefully counteract the bias of leaning to one threshold. As an example, if the threshold is set to 0.2, then all K recommendations will be compared with the user profile, and if there is a recommendation that has 20% of the features that a user profile recipe has, then the recommendation is labelled as 'relevant' to the user.

With regards to relevance levels for the NDCG@K score, there are a set of thresholds defined which map to a level of relevance. For example, with the threshold set [0.2, 0.25, 0.3], the relevance levels are mapped to 1, 2 and 3 respectively. So, the NDCG@K score labels a recommendation with a level of relevance, depending on which percentage threshold range it falls under.

Plots will represent a test with a type of user and parameter values that are set. 'Rated'/'rating' represents the number of recipes rated by the user, 'limit' represents the user profile size and 'threshold' represents the set of threshold values for relevance. Each test will contain three AP@K plots as I wanted to show three different perspectives for how relevance can be defined for the score. Lastly, 'with ncf' represents the Hybrid model and 'without ncf' represents the CBF model without the influence of the NCF.

Limitations:

Finding the AP@K scores for each threshold is a lengthy process, and so, there are plots that do not include the NDCG plot, as this was a metric that was decided on after the experiment was conducted. Due to the time taken to conduct the experiment, I did not redo them with the NDCG plot. I have also had to limit the number of tests due to the computation time and have separated out tests in consecutively even number of user profile recipes. Lastly, any names used here was randomized, and are not the names of the actual users from the food.com dataset.

1. Sampling random users

To start off, I sampled 10 users with at least 3 ratings.

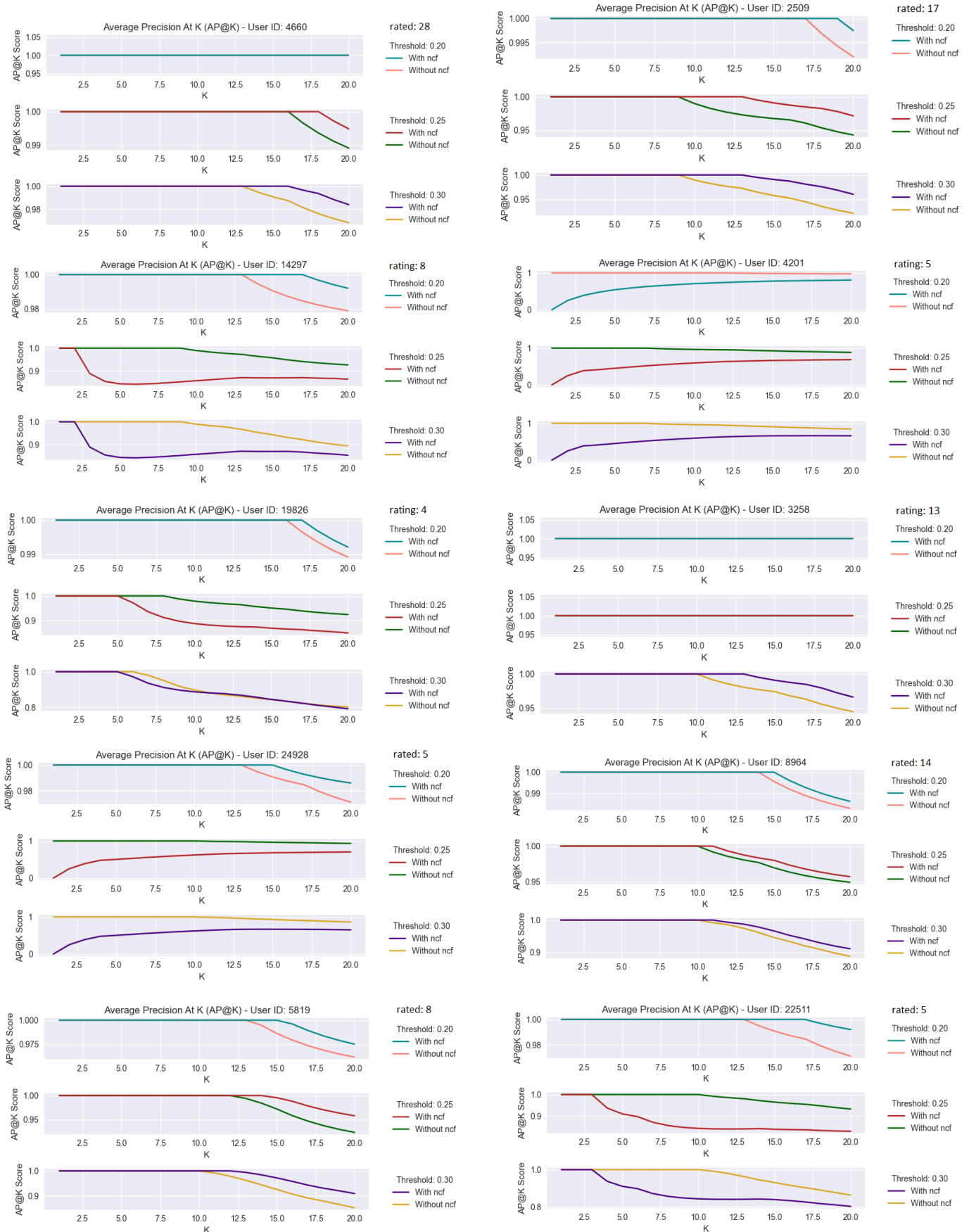


Figure 11. A set of plots showing how the AP@K score varies as K increases for 10 users.

From these results, we can see that for users with $\sim \leq 8$, the AP@K score is greater without using the NCF model. For users with > 8 ratings, the AP@K score performs consistently better with the NCF model. With these results, we can see that the size of a user's profile greatly influences the performance of the CBF combined with NCF, and that, the Hybrid model favours users with more ratings.

2. Ranging the size of the user profile for a single user

Six tests to explore the impact of evaluation scores, using various profile sizes.

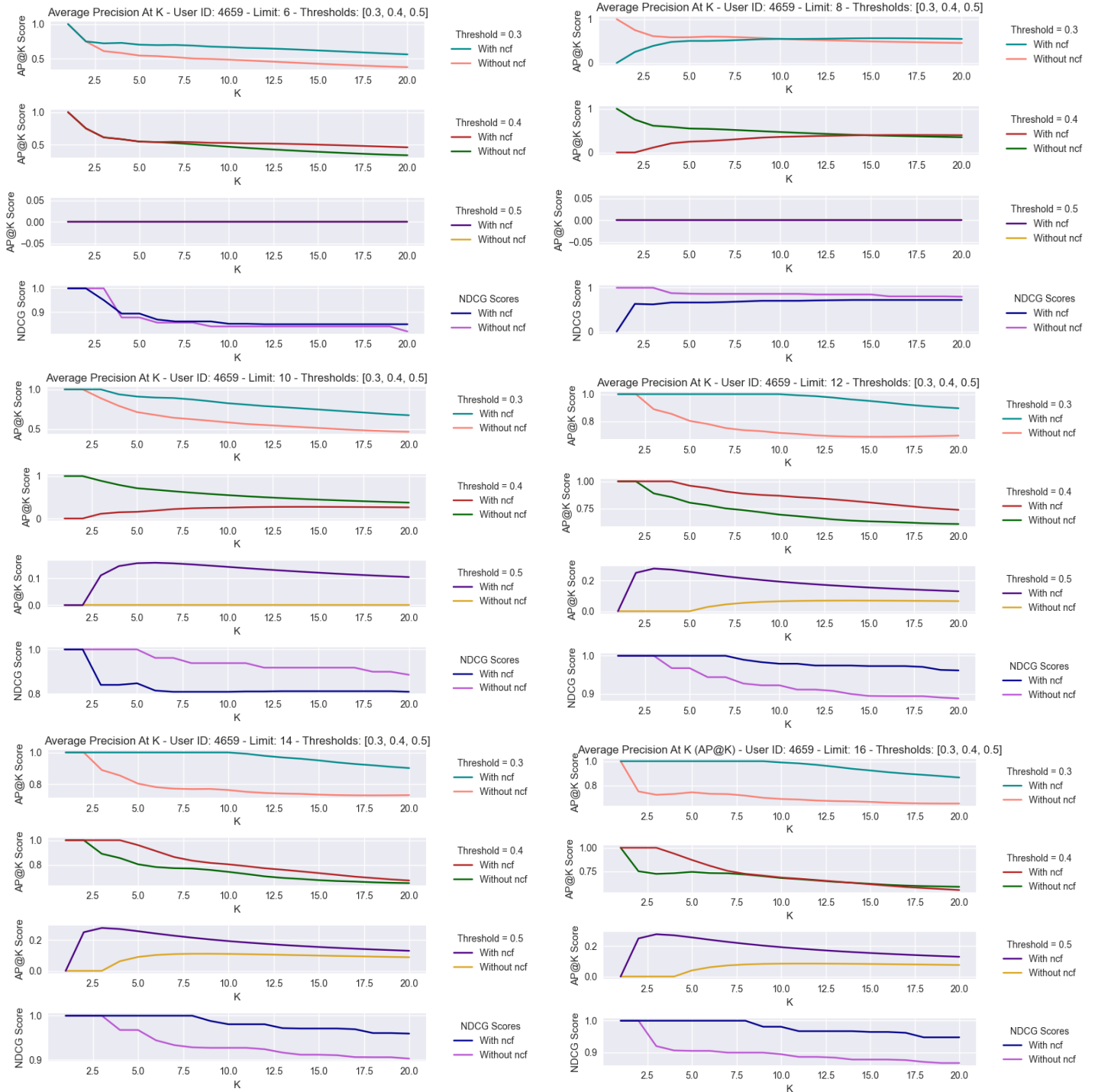


Figure 12. Plots showing how AP@K and NDCG@K varies for different user profile sizes.

From these results, it is evident that the same sentiment is shared with experiment 1 – as the number of recipes included in the user profile increases, the greater the performance of the Hybrid model, when compared to the CBF. On the other hand, the CBF performs better when there are less than 10 ratings which could mean that there not enough ratings to gauge the preference of similar users. As a sidenote, at the time of conducting this experiment, I increased the threshold for the percentage of features that must be common between a user profile recipe and a recommendation, as I felt that I could be stricter with how I define the boundary of how relevant a recipe is.

3. Sampling users with a single rating

Evaluating the models with users that have one rating.

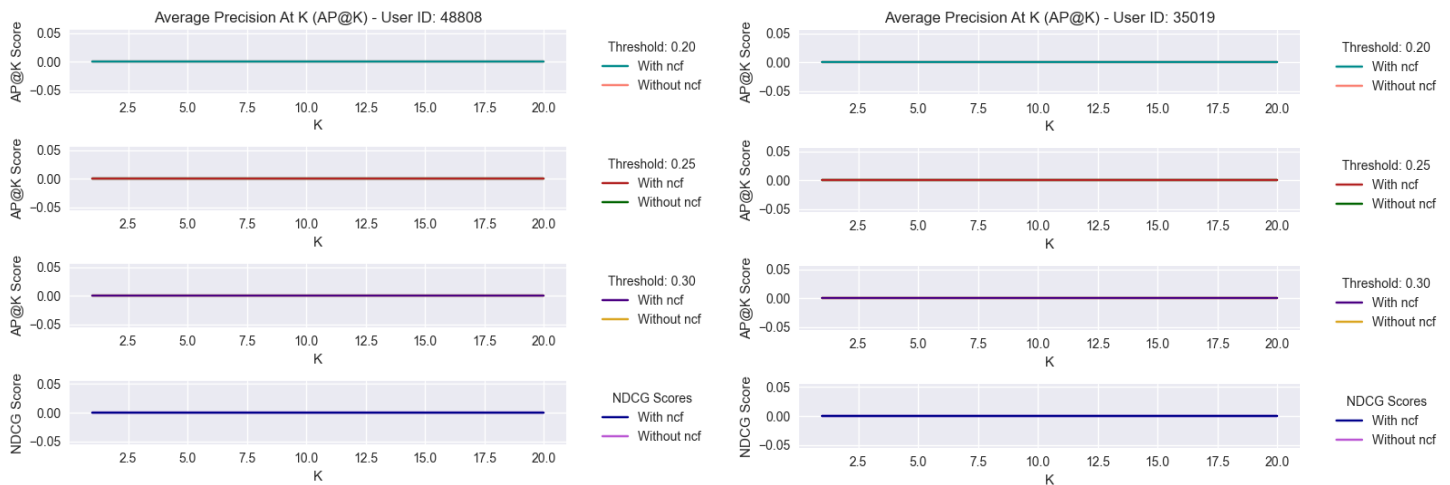


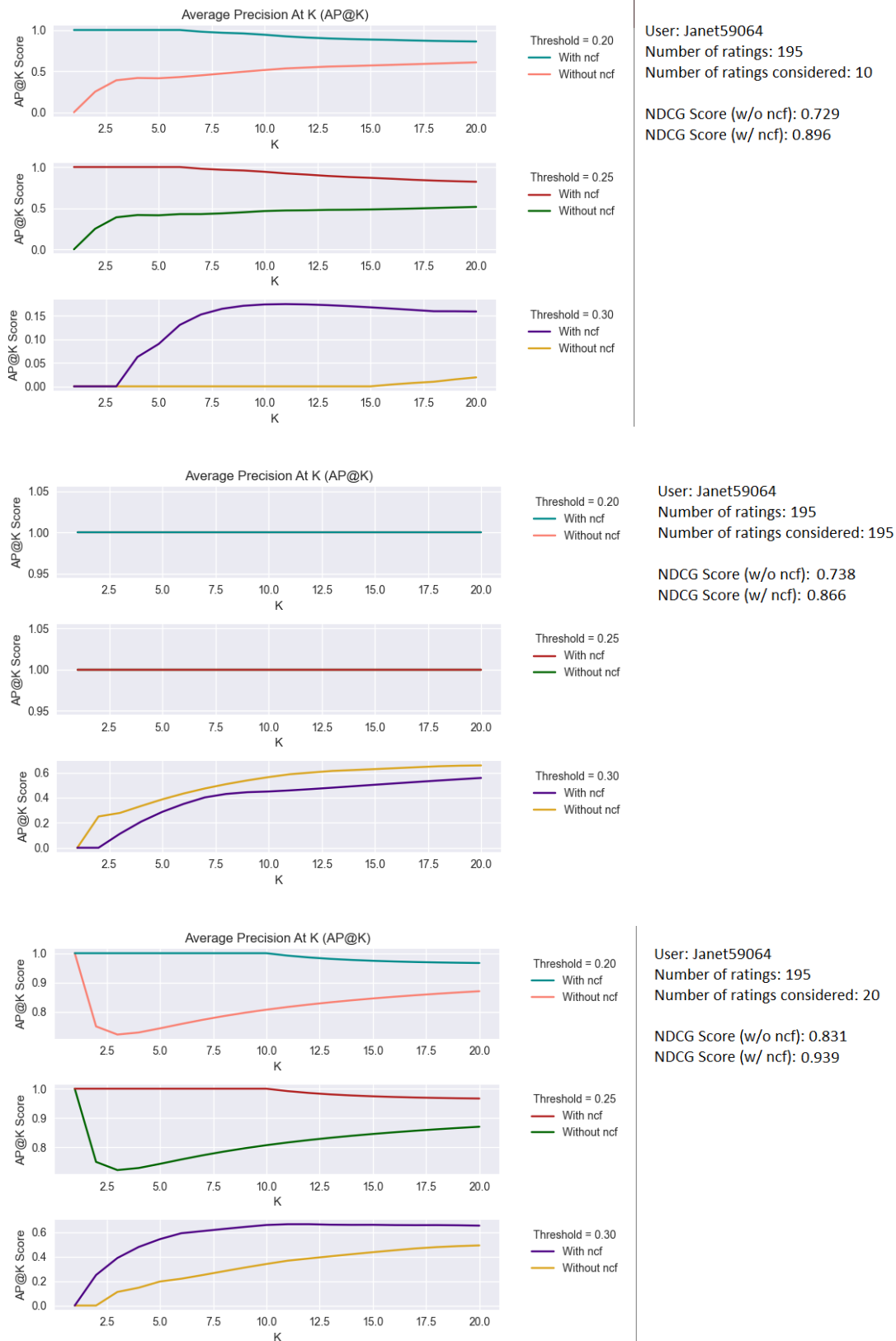
Figure 13. Shows how AP@K and NDCG@K varies for two users that have only rated one recipe.

As expected, the Hybrid and CBF cannot produce a sufficient profile of a user with only a single recipe to base it off, hence 0% AP@K and NDCG@K scores.

4. Finding a suitable size for the user profile

Testing with two types of users, both with a very high number of recipes rated.

Firstly, testing extremes in user profile size for user 1: Janet.



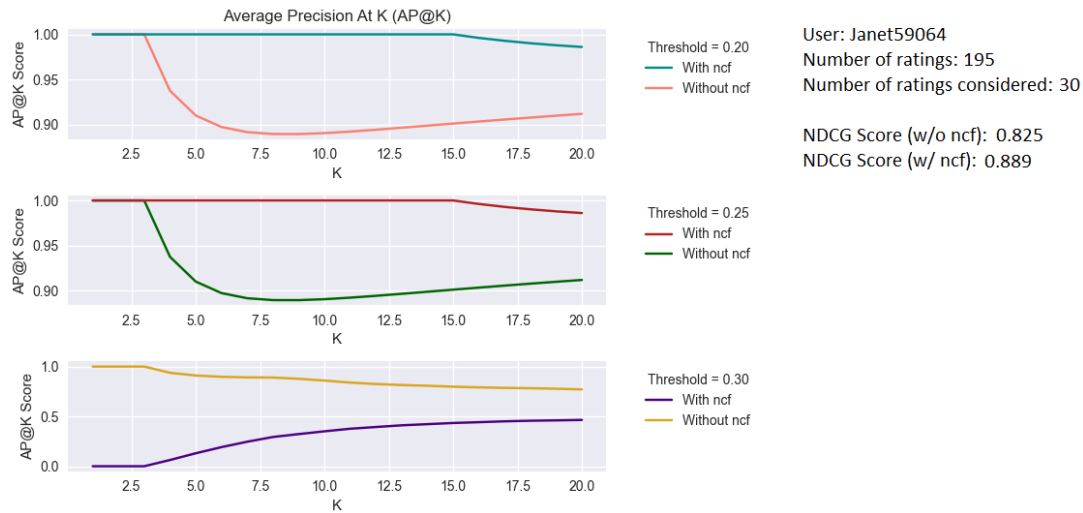


Figure 14. Plots for Janet showing how AP@K varies for different user profile sizes.

With Janet, I wanted to see the impact on the user profile when testing the extreme ends, 10 and 195. As expected, the Hybrid model performed with 100% AP@K score when considering 95 recipes for the user profile, which suggests that most recipes that are recommended would be ‘relevant’ to the user – this is logical, as there would be such a large pool of recipes considered in the user profile that there is bound to be a recipe that is relevant for the user at every precision level. Considering only 10 recipes for the user profile, produces great performance when considering the NCF model as opposed to not considering the NCF model, without any over influence in both curve plots that was evident when 195 ratings were considered, which is consistent with experiments 1 and 2.

I then added a couple of extra plots with 20 and 30 ratings considered, to see the influence on the AP@K score and the user’s mean NDCG@20 score. 30 ratings show there is over-influence from the user profile, which is why there is almost 100% in thresholds 1 and 2, just as we saw with the 195 sized user profile. And again, without the NCF performed better, which could mean that there aren’t many similar users that have rated as many or, the same kinds of recipes as Janet. Size 20 is consistent with what we saw in previous experiments, and, resulted in the best NDCG@20 score of 94%.

Increasing the user profile size for user 2: Tristan.

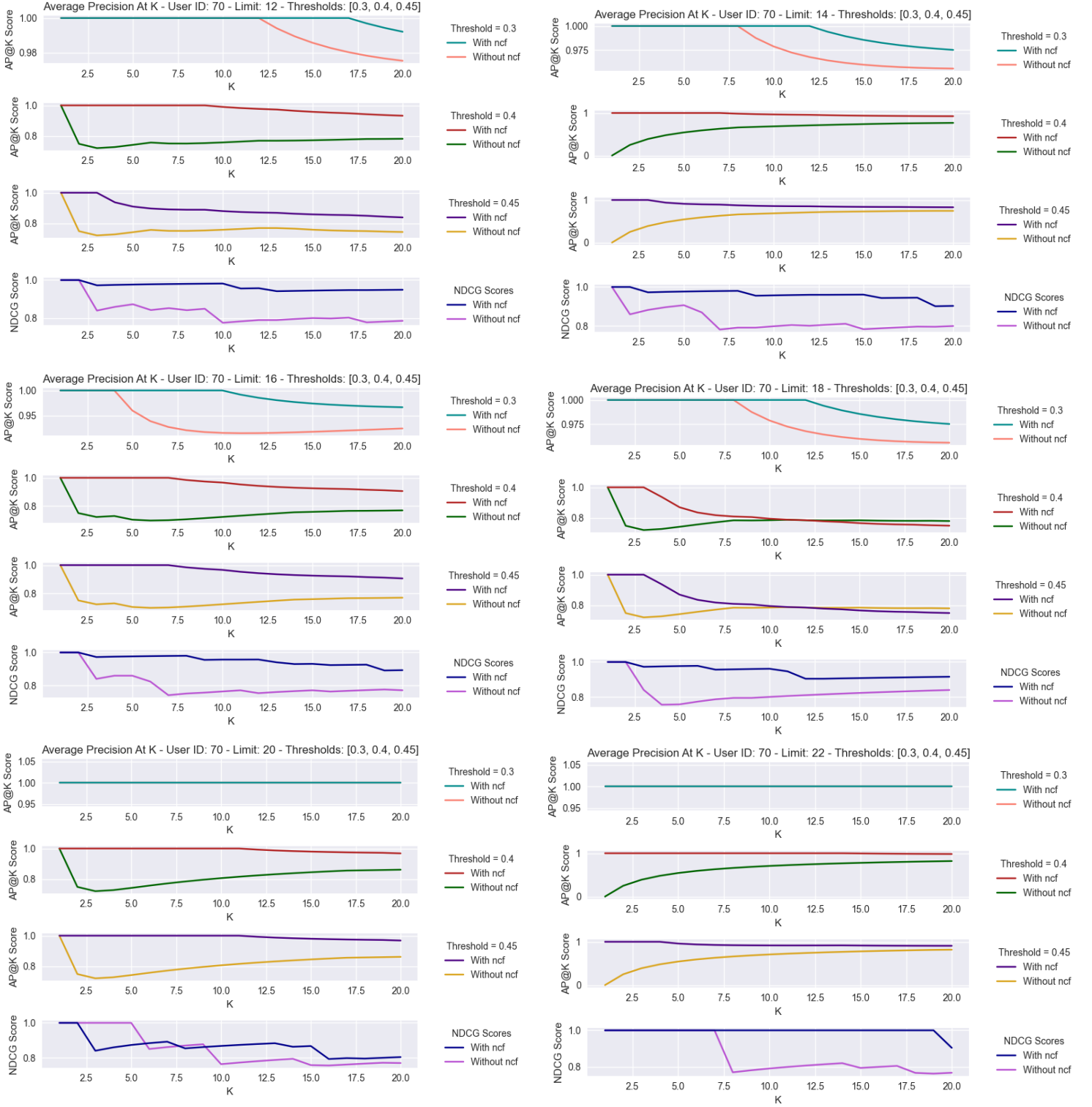


Figure 15. Plots for Tristan showing how AP@K varies when increasing the user profile sizes.

With Tristan, we see that as the number of recipes considered in the user profile increases, there is more and more influence in the AP@K score. In this case, limiting the threshold to a size of 16-18 results in a reasonable influence in the AP@K score, where it is not converging to 100% as with limits 20 and 22.

To conclude with these two tests, 16-20 is within the realm of an ideal user profile size, which is consistent with prior experiments.

5. Varying threshold values

Evaluating with various threshold configurations.

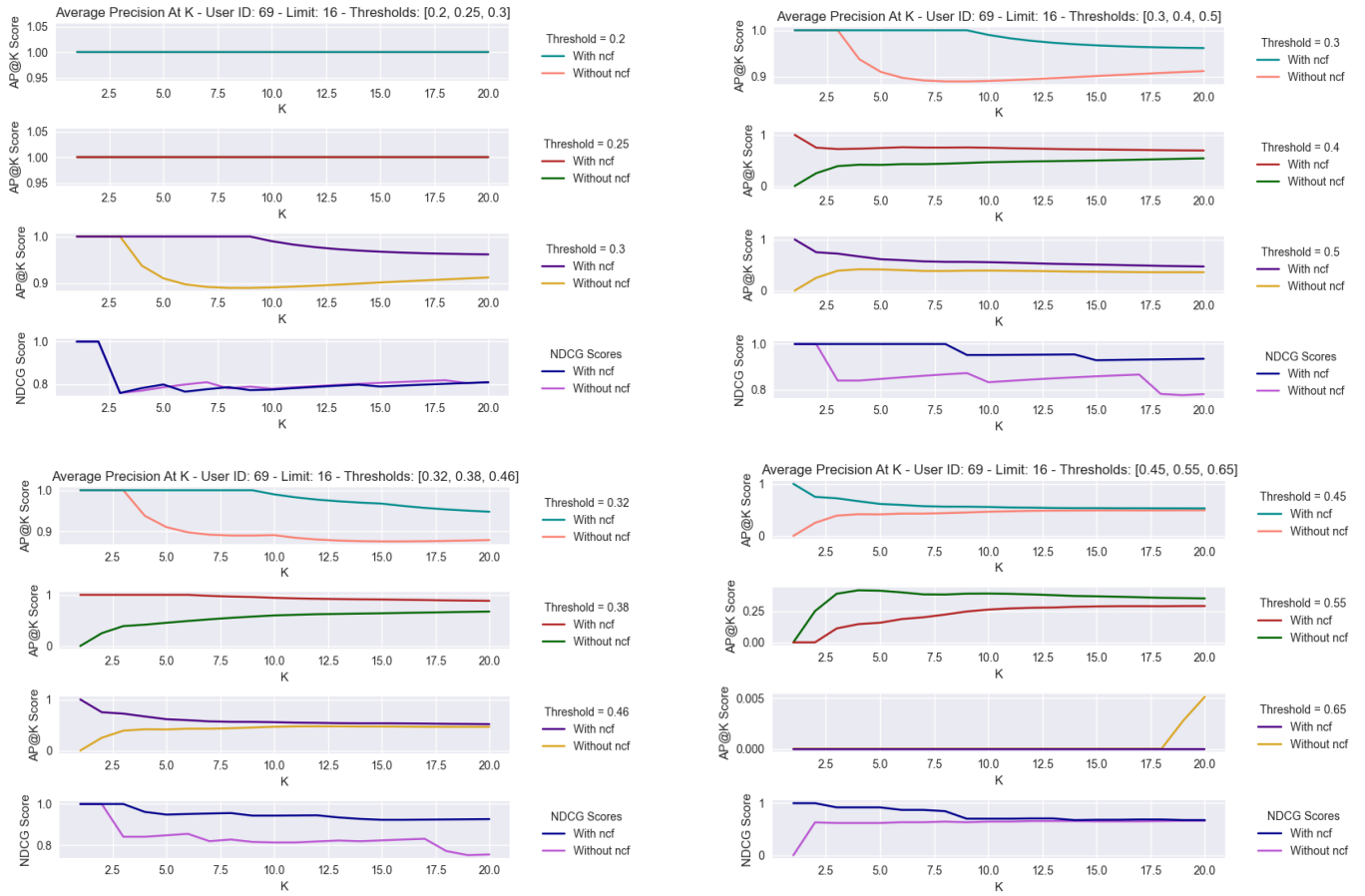


Figure 16. Showing how AP@K and NDCG@K vary as the threshold configurations increase in value.

Thresholds selected are mostly equidistant, and they rise in each successive test whilst keeping the same user being evaluated on and the same profile size. In test 1, we see the AP@K score is either 100% or close to 100%, which shows that it is very common for recommendations to share 20-25% of features for a recipe this user has rated in the past.

In test 4, we see that there are roughly the same number of recommendations that share 55% with a recipe from the user profile, but hardly any share 65%. The NDCG score also becomes equivalent whether the NCF is included or not. Therefore, these threshold values are too extreme and do not provide enough nuance for defining relevance levels. The threshold levels from test 2 will be used for the next two experiments.

6. Testing users that have a high mean rating

Evaluating on a user that tends to score very highly.

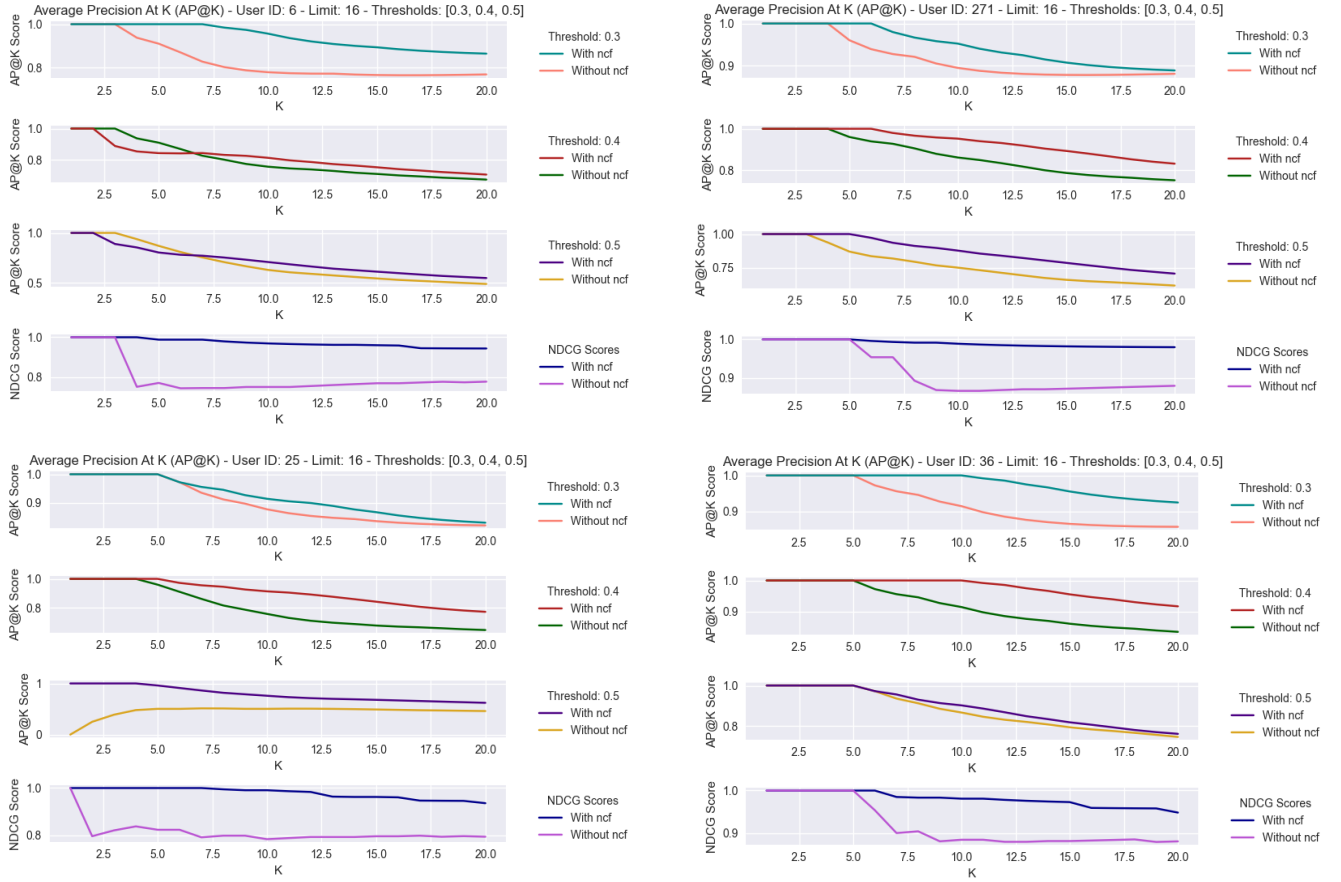


Figure 17. Plots for users with an average rating score greater than 4.

From these results we see that for all users, the Hybrid model outperforms the CBF in AP@K and NDCG@K scores when $K > 7$ (on average). In a couple of cases, the Hybrid model is almost comparable to the CBF.

7. Testing users that have a low mean rating

Evaluating with users that tend to rate very harshly.

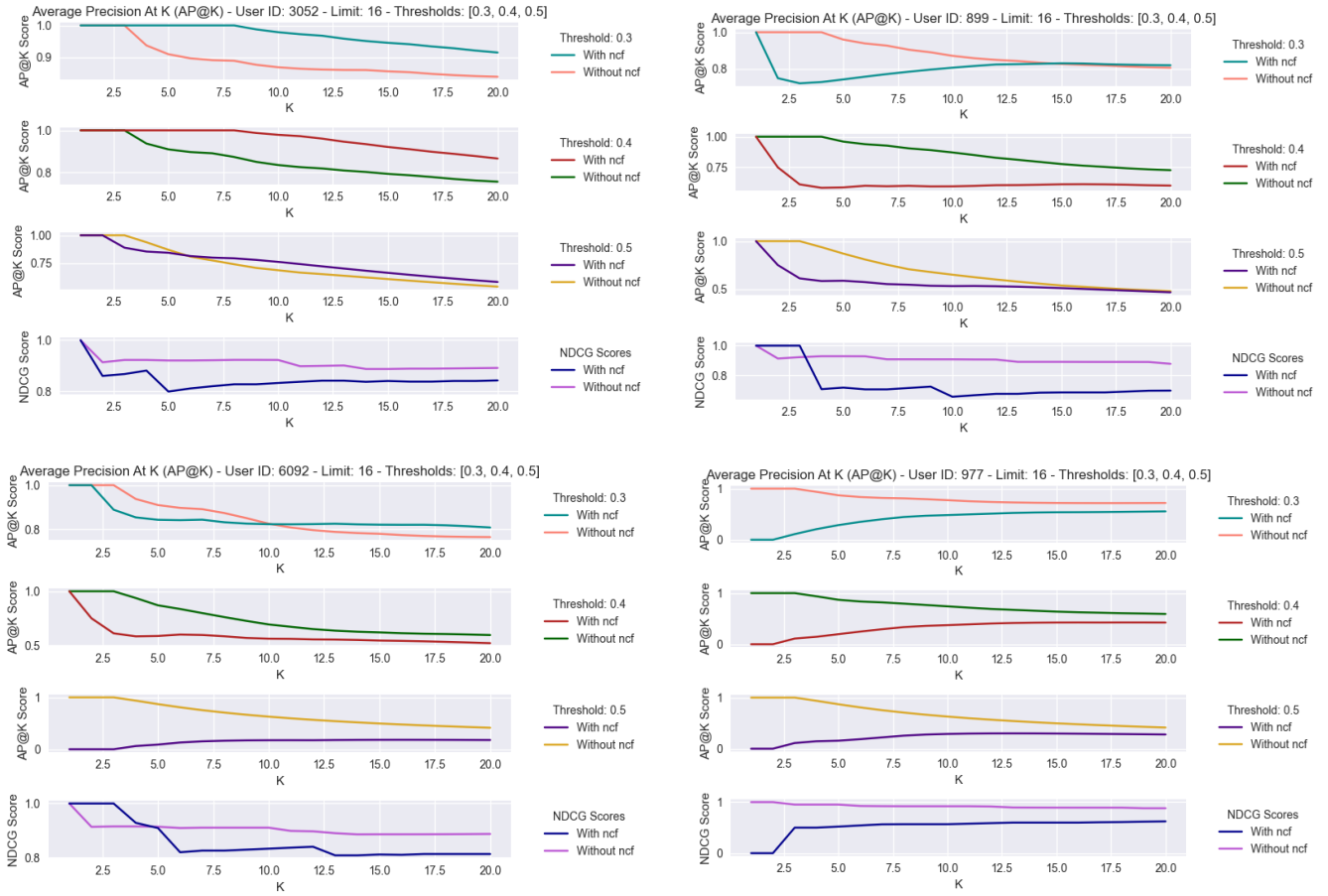


Figure 18. Plot for a user with an average rating score lower than 1.5.

Out of 68325 users on the website database, there were only 4 users that have an average rating score of 1.5 with at least 16 ratings. In figure 14, for the most part, we see that the Hybrid model consistently performs worse when compared to the CBF model. This may be due to there being a very low percentage of users with as low of a mean as 1.5 with the same sort of recipe preferences. Therefore, the NCF model was unable to use similar user preferences to find recipes that are most relevant/suitable to the user. With a lot more users on the system, I would expect the NCF model to perform adequately.

5.2 Final Results

Using a validation set of 1000 users, with threshold values [0.3, 0.4, 0.5], limit=16, with users sampled with at least 16 ratings and K=20, the Mean Average Precision (MAP) (threshold = 0.4) and Mean Normalized Discounted Cumulative Gain (MNDCG):

Hybrid: MAP = 79.65%, MNDCG = 84.22%

CBF: MAP = 68.10%, MNDCG = 71.42%

These results suggest that across the board, the Hybrid model provides highly relevant recommendations to users and consistently outperforms the CBF model due the insight that is gained from similar users.

Considerations

These results are good but there are a few compromises to consider.

Firstly, it is clear from experiment 2 that a user needs at least 10 ratings on the website for the NCF model to greatly influence recommendations. So, this tells us that the merging of the NCF and CBF does not help with solving the cold start problem – we saw in chapter 2 that this was the downside to collaborative filtering. This was not the goal for the Hybrid model, but these results highlight that the preferences from similar users cannot be gauged for a user with low number of ratings, and that it is not improved upon by considering the content of each recipe. This was instead the goal of the PR. We see that the CBF model alone provides more relevant recommendations to the user, which is much better for addressing the cold start problem. Henceforth, the CBF recommender will be used for recommendations if the user has less than 10 ratings.

Secondly, the Hybrid model performs worse than CBF when users have a mean rating lower than 1.5. This is most likely due to the population of users with a mean rating less than 1.5 being incredibly small, and so, the NCF model is unable to gauge the preferences of similar users. In cases where a user has a low mean rating, the CBF will be used instead.

Lastly, the Hybrid model rests on the assumptions that the user does not want to try a recipe highly related to a recipe they have previously rated with a low score (below their mean rating). In chapter 4, I mentioned that the candidate recipes for the user profile are those that the user has rated most recently and is above their mean. If the user does not have at least 10 ratings due to this candidate item selection process, then the CBF model is used instead, due to the reasons in the first point.

6. Conclusion

Limitations & future improvements

Implementation of the website and the models came with some limitations that are worth mentioning.

In the 'Recipes' dataset, the tags that were used were not diverse enough. Many recipes share the same tag and so completely unrelated recipes would have the tag features in common – tags such as 'cuisine', 'ready under 60 minutes' and 'homemade'. The tags are not specific enough for a recipe, which means that critical information that can be used to see if two recipes are similar to each other is lost. More descriptive and specific tags could improve the ability for CBF to decide whether two recipes are similar.

Another downside to the lack of diverse hashtags, is with the user experience on the website. Just as Cooking New York Times did, recipes could be organized in meaningful collections such as 'Healthy Breakfasts', 'Kids Dinner' and 'Vegetarian Cuisines' – recommendations could be given with recipe collections along with individual recipes.

Due to the memory needs of the CBF, the recipe dataset had to shrink by a factor of 8 in order to stay at a reasonable size for use and evaluation. With access to more memory, a wider range of recipes can be compared with each other and the recommendation set can show more relevant recipes. It would also mean the number of users in the database would increase, which means there will a larger pool of users that are similar to each other, which can improve NCF performance.

Websites like Kitchen Stories have their own selected editors making recipes, which means KS has control over the content in each recipe, and so ultimately, they can produce better similarity measures.

For the future, I would incorporate a knowledge-based recommender system which would ask a series of questions to a new user, to gauge their personal preferences and then build a knowledge profile. A blend of this profile and the user's current rating profile would be the basis for recommendations until the user has rated 'enough' recipes on the website, where the Hybrid model will solely be relied on. This methodology provides the system with more information about the user to better personalize the user's experience in the early stages, hence this would be another way to address the cold start problem.

I would also want to find a dataset with recipe data that is more diverse, or I would want to produce my own data by merging datasets and generating tags.

For deployment purposes, I would use Cloud Computing to have access to more memory, and I would use a Rest API to call my models.

Overall conclusion

Models performed consistently well on the experiments conducted, and on the validation set the final metric scores were: MAP = 79.65% and MNDCG = 84.22%. Evaluation on the Hybrid model showed that the influence of the NCF caused the recommendations to show less relevant recipes to a user with a very low mean, due to there not being enough similar users in the database. In all other experiments, the Hybrid model improved the quality of recommendations over the CBF, given they have rated at least 10 recipes.

References

- [1] Pasquale Lops, Marco de Gemmis, Giovanni Semeraro, "Content-based Recommender Systems: State of the Art and Trends", in *Recommender Systems Handbook* (pp.73-105) (2011) DOI: https://doi.org/10.1007/978-0-387-85820-3_3
- [2] B. Stecanella, "What is TF-IDF?", [Online, accessed on: 22/11/2020], URL: <https://monkeylearn.com/blog/what-is-tf-idf/>
- [3] D.M. Pennock, E. Horvitz, S. Lawrence and C.L. Giles, "Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach", Jun 2000.
- [4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T-S Chua, "Neural Collaborative Filtering". In *Proceedings of the 26th International Conference on World Wide Web 2017*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 173–182. DOI: <https://doi.org/10.1145/3038912.3052569>
- [5] G.O. Singer, and A. Sundararajan, "Recommendation Networks and the Long Tail of Electronic Commerce.", *MIS Quarterly* 36, no. 1 (2012): 65-83.
- [6] J. Jitchotvisut, "The hottest food trend in the year you were born", Nov 2018. [Online, accessed on: 19/11/2020], URL: <https://www.insider.com/most-popular-food-trend-each-year-2018-9#2018-isnt-over-yet-but-so-far-mealprep-seems-to-be-hitting-its-stride-across-the-internet-if-not-the-country-64>
- [7] Social Films, "TikTok UK Statistics 2020". [Online, accessed on 19/11/2020], URL: <https://www.socialfilms.co.uk/blog/tiktok-uk-statistics#:~:text=TikTok%20Demographics%20UK,-So%20who%20exactly&text=In%202019%2C%20the%20largest%20age,users%20accessing%20TikTok%20from%20smartphones.&text=Only%20a%20combined%20total%20of,of%20the%20app%20last%20year>
- [8] P. Leskin, "TikTok surpasses 2 billion downloads and sets a record for app installs in a single quarter", Apr 2020. [Online, accessed on 23/11/2020], URL: <https://www.businessinsider.com/tiktok-app-2-billion-downloads-record-setting-q1-sensor-tower-2020-4?r=US&IR=T>
- [9] M. Mohsin, "10 Tiktok Statistics That You Need To Know In 2020", Sep 2020. [Online, accessed on 19/11/20], URL: <https://www.oberlo.co.uk/blog/tiktok-statistics>
- [10] J. Alexander, "TikTok reveals some of the secrets, and blind spots, of its recommendation algorithm", Jun 2020. [Online, accessed on 19/11/2020], URL: <https://www.theverge.com/2020/6/18/21296044/tiktok-for-you-page-algorithm-sides-engagement-data-creators-trends-sounds>
- [11] J. Howard, R.Thomas, S.Gugger, "fastai" published by GitHub, 2018. URL: <https://docs.fast.ai/collab>
- [12] B.P Majumder, S. Li, J. Ni, J. McAuley, "Food.com Recipes and Interactions, Crawled data from Food.com (GeniusKitchen) online recipe aggregator", 2019. [Accessed on 22/11/2020], URL: <https://www.kaggle.com/shuyangli94/food-com-recipes-and-user-interactions>
- [13] B. Stecanella, "What is TF-IDF?", May 2019. [Accessed on 15/01/2021], URL: <https://monkeylearn.com/blog/what-is-tf-idf/#:~:text=TF%2DIDF%20is%20a%20statistical,across%20a%20set%20of%20documents.>
- [14] S. Glen, "Seasonal Kendall Test – Statistics How To", May 2016. [Accessed on 03/02/2021], URL: <https://www.statisticshowto.com/seasonal-kendall-test/>
- [15] M. Malaeb, "Recall and Precision at K for Recommender Systems", Aug 2017. [Accessed on 9/02/2021], URL: https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

- [16] J.L Herlocker, J.A Konstan, L.G Terveen, J.T Riedl, "Evaluating Collaborative Filtering Recommender Systems". In *ACM Transactions on Information Systems*, Vol. 22, No. 1, January 2004. DOI: <https://doi.org/10.1145/963770.963772>
- [17] G. Shani, A. Gunawardana, "Evaluating Recommendation Systems", in *Recommender Systems Handbook* (pp 257-297) (2011), DOI: https://doi.org/10.1007/978-0-387-85820-3_8
- [18] S. Sawtelle, "Mean Average Precision (MAP) For Recommender Systems", Oct 2016. [Accessed on 02/15/2021], URL: <https://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html#:~:text=In%20recommendation%20systems%20MAP%20computes,relevant%22%20recommendations%20for%20that%20user>
- [19] P. Chandekar, "Evaluate your Recommendation Engine using NDCG", Jan 2020. [Accessed on 03/15/2021], URL: <https://towardsdatascience.com/evaluate-your-recommendation-engine-using-ndcg-759a851452d1>
- [20] K. Pykes, "Normalized Discounted Cumulative Gain", Aug 2020. [Accessed on 03/15/2021], URL: <https://towardsdatascience.com/normalized-discounted-cumulative-gain-37e6f75090e9>
- [21] Whirlpool Corp, "Yummly: Personalized Recipe Recommendation and Search", 2021. [Accessed on 10/03/2021], URL: <https://www.yummly.co.uk/>
- [22] Django Software Foundation, "The Web framework for perfectionists with deadlines", 2021. [Accessed on 08/03/2021], URL: <https://www.djangoproject.com/>
- [23] M. Hudelson, "Django's Architectural Pattern | The Turning Gear", Oct 2018. [Accessed on 08/03/2021], URL: <https://www.theturninggear.com/2018/10/22/djangos-architectural-pattern/>
- [24] J. Madie, "LibRecommender 0.6.4", Apr 2021, *Licensed by MIT*. [Accessed on 10/4/2021], URL: <https://pypi.org/project/LibRecommender/>
- [25] J. Madie, "LibRecommender 0.6.4", Apr 2021, *Licensed by MIT*. [Accessed on 12/4/2021], URL: <https://github.com/massquantity/LibRecommender/tree/master/examples>
- [26] J. Moody, "What does RMSE really mean?", Sep 2019. [Accessed on 12/4/2021], URL: <https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e>
- [27] W.L. Rice, T.J. Mateer, N. Reigner, "Changes in recreational behaviors of outdoor enthusiasts during the COVID-19 pandemic: analysis across urban and rural communities", in *Journal of Urban Ecology*, Volume 6, Issue 1, 2020. DOI: <https://doi.org/10.1093/jue/juaa020>