Anime Suggestion System

Lisa Ye, Iris Li, Anubha Joshi, Simran Vig

April 2023

Introduction

Anime, Japanese animations that are typically aired episodically in seasons, have been growing more and more popular in recent years. Episodes are often short and dense, making them oftentimes bingeable—which, however great, is accompanied with the empty feeling of great shows ending too quickly. It can be difficult to find new anime to watch after recently finishing a show, especially when searching for something good to fill that gaping void—seeing as there are hundreds of shows to choose from and thousands of differing opinions from viewers on their quality. As such, **our goal was to generate a list of anime that the algorithm's user is most likely to enjoy, based on the preferences that they input**. To do this, we needed to create a recommendation system.

A recommendation system is an AI algorithm that predicts user preferences and suggests items that one would be more inclined to like, buy, or interact with (Shetty). Often seen in product suggestions on e-commerce platforms—"you might also like…" pages—or highly personalized "for you" pages on social media, recommender systems are powerful, now commonplace, tools that allow us to predict user behavior, and capitalize on it.

There are a variety of different recommender systems; many of which filter data based on the relationships between user interactions with items, personal preferences, and user demographic. Collaborative filtering, for example, predicts your interests by identifying the preferences of users with similar tastes (Vatsal). The principle, in the context of anime recommendations, would be that if Alice and Bob have similar tastes in anime, then Bob will be likely to enjoy the anime Alice enjoys, and vice versa. Content-based filtering on the other hand, would focus on matching you to items based on your personal history, such as anime you've previously enjoyed (Vatsal). Figure 1.1 illustrates these concepts.

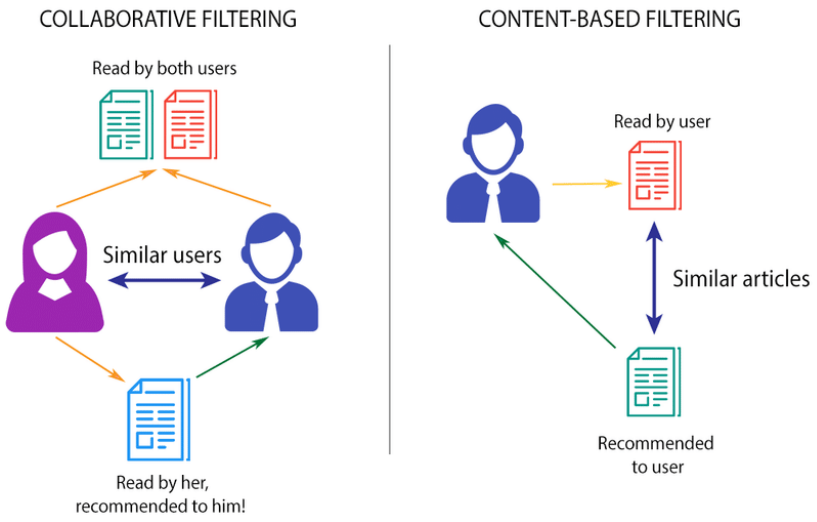

Figure 1: Contest Based Filtering vs Collaborative Filtering (Research Gate 2018)

Alternatively, there are hybrid models that combine the underlying ideas of user similarity with collaborative filtering, and the user's evaluation history with content-based filtering (such as similarity-based recommendation.)

Our algorithm is a hybrid model which uses some principles of collaborative filtering and similarity based recommendation. In addition to returning a list of recommended anime, it returns a graph which shows the 'cluster' that the user belongs to, and the connection between the outputted animes and other users' preferences. This is explained in more detail in our computational overview.

Datasets

We decided to use two existing datasets which are originally from Kaggle's database and manipulate them into our own CSV files. Kaggle is a widely used data science website which includes datasets across many areas and is highly recommended by computer scientists and data scientists alike.

The information in both datasets is taken from My Anime List, a website which anime-lovers from around the world use to keep track of and rate the animes they've seen, and post reviews for other users to see.

The first dataset, titled MyAnimeList Dataset, has a file called UserAnimeList.csv which includes information about users and how they rated different animes on the My Anime List website.[1] The second dataset, titled Anime Dataset with Reviews - MyAnimeList, has a file called animes.csv, which includes information about the animes on My Anime List including their ids and the genres that

they belong to.[2] We split these datasets into multiple CSV files to refer to in our code. Therefore, the samples of each dataset provided below are also in CSV format. These files are called 'animes.csv,' 'genres.csv,' and 'users.csv.'

The first dataset includes information about users on the website and the animes that they have seen. This data was used to create users.csv.

Three columns from the original dataset were used to create the users.csv file. These are the columns which include the user's username, the title of an anime they've watched, and their rating of the show from 0 to 10, which refer to columns 0, 1, and 5 of the original file from the first dataset respectively. Here is a sample line of data from the users.csv file:

Kerma_, Shingeki no Kyojin, 5

The second dataset includes information about animes and their corresponding genres. Only two columns from the initial dataset, 1 and 2, were used to create the animes.csv and genres.csv files. These columns include the id of each anime (a multi-digit integer) and their corresponding genres.

A row in the animes.csv file includes the title of an anime, the first of its genres, the second, the third, etc. Here is a sample line of data from this file:

Haikyuu!! Second Season, Comedy, Sports, Drama, School, Shounen

A row in the genres.csv file includes the title of a genre, the first title in that genre, the second, the third, etc. Here is a sample line of data from this file:

Comedy, Haikyuu!! Second Season, Fullmetal Alchemist: Brotherhood, Mob Psycho 100 II, Owarimonogatari 2nd Season, Haikyuu!!: Karasuno Koukou vs. Shiratorizawa Gakuen Koukou, ...

The ellipsis represents more animes that correspond to this genre but are not included here for the sake of space.

Computational Overview

Using some principles of collaborative filtering and similarity-based recommendation from our research (Kurama), we decided on the following approach: get a rating (or potential rating) mapping between a set of users and animes, and convert that to a bipartite graph.

Using this mapping, we can use similarity-based recommendations based on the user of the program, and community detection to find similar users and output their top-rated animes (Vastal).

Community detection is a method used to group nodes into 'clusters' based on specified factors. Our algorithm uses the Louvain method for community detection, which is a technique to extract and generate clusters from large networks. It employs modularity, a community detection optimization method which measures the density of connections, or edges, between nodes in a cluster. The Louvain method employs this idea of modularity by accounting for the density and strength of connections in the clusters and uses a greedy optimization method for modularity (Lancichinetti & Fortunato). Its algorithm is beyond the scope of CSC111, however, as it calculates modularity based on the weights of edges, and accounts for the communities of the nodes with the Kronecker delta function (Ibid).

In this way, graphs play a central role in our algorithm. NetworkX is an external library which allows us to visualize these graphs and utilise Louvain's community detection. Figure 2 illustrates the concept of community detection.

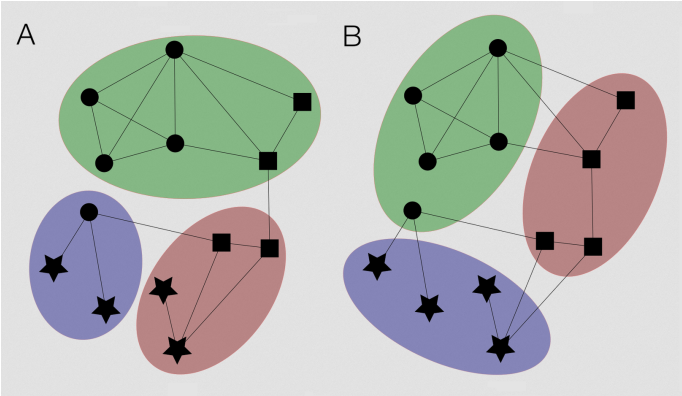

Figure 2: Community Detection (atlas.brussels)

It is predicted that similar users with similar tastes in animes that they have already watched would have similar preferences in animes they have not watched yet (Shetty). This provides the basis for why our recommendation system works.

Here are the steps that outline our code's computations:

Initial Processing:

1) Using our Kaggle dataset for user ratings[1] and animes and genres[2], our extract_data function creates three CSV files that we will use in our algorithm. These files are animes.csv, users.csv, and genres.csv

2) In the Data class, found in the data_class.py file, there are three functions which create useful mappings. These are find_anime_to_genre, which outputs a dictionary mapping of anime titles to their corresponding genres, and

find_genre_to_anime, which outputs a mapping of genre titles to their corresponding animes. The function find_user_to_rating outputs a mapping of usernames to another dictionary mapping of the titles of animes the user has watched to their rating of the show. These mappings are initialized as instance attributes as an instance of the Data class. Here are examples of each attribute:

anime_to_genre:

{'Haikyuu!! Second Season': Comedy, Sports, Drama, School, Shounen}

genre_to_anime:

{Comedy: 'Haikyuu!! Second Season', 'Fullmetal Alchemist: Brotherhood', 'Mob Pyscho 100 II', 'Owarimonogatari 2nd Season', 'Haikyuu!!: Karasuno Koukou vs Shiratorizawa Gakuen Koukou', ...}

user_to_rating:

{Xyik: {'Attack on Titan': 9, 'Hunter x Hunter': 10}}

3) In the calculations.py, there are two functions which calculate data that we later use to create two matrices. The first is user_to_genre_compatibility, which outputs a float from 0.1 to 1.0 (corresponding to a scale from 1 to 10) that represents a user's affinity to a certain genre. The function averages a user's ratings for the shows that they have watched in a certain genre.

The second function is anime_to_genre_compatibility, which outputs a float that represents the compatibility from 0.1 to 1.0 that the given anime has to the given genre. The function multiples the user's ratings for the given show by their user_to_genre_compatibility for the given genre, and takes an average of this for all users. If some users have not seen certain shows from this genre, a default rating of 0.5 will be inputted. Additionally, we reward or penalize an anime on their compatibility to the genre based on whether it falls under the genre in question. If it does, the 8th root of the calculated average is returned, otherwise it is raised to the 8th power and returned. The number 8 was decided upon by trial and error with various different values.

4) Then, in the matrices.py file (which inherits from calculations.py), the above information is used to create two matrices, and then multiply them. The first matrix is created in the create_user_to_genre_matrix function. This function uses the user_to_genre_compatibility function to create a matrix mapping of all users in the given Data instance to another dictionary which maps genre titles to the user's compatibility to each genre.

The second matrix is created in the create_genre_anime_matrix function. This function uses the anime_to_genre_compatibility function to create a matrix map-

ping of all animes in the given Data instance to their compatibility with each genre in the given Data instance.

The mat_mul_map function then takes these two matrices and performs matrix multiplication. The output is a dictionary which maps a username to another dictionary, which maps anime titles to the user's predicted enjoyment of the anime from 0.1 - 1.0. These float values represent the weights of the edges between the nodes (anime titles and usernames) in our bipartite graph. Figure 3 illustrates the idea of matrix multiplication.

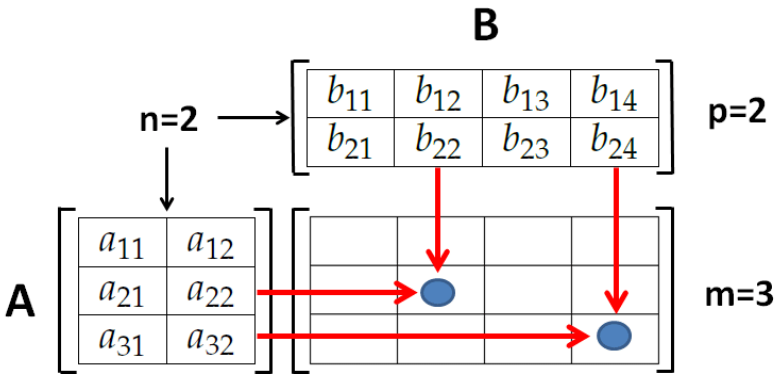

Figure 3: Matrix Multiplication (Research Gate 2013)

Creating the Graph:

Using the dictionary from the mat_mul_map, we used NetworkX to populate our graph, with the populate_graph function. This function uses built-in functions from the NetworkX library such as add_node and get_node_attributes. This function works in the following ways:

1) First, it instantiates a NetworkX.Graph variable.

2) Then, it iterates over every user key in the dictionary from mat_mul_map and adds it as a node with 'type' attribute 'user.'

3) Within that iteration, there is a nested iteration for every anime, where the anime title is added as a node with 'type' attribute 'anime.'

4) Using the dictionary from mat_mul_map, the function sets a weight attribute, which is the user to anime compatibility value, for the edges between the anime and user nodes.

5) After all the iterations are complete, populate_graph returns the bipartite graph.

User Input Stage:

1) After running the main.py file in the console, the client will be prompted to input the title of an anime they've watched. This anime must be within our datasets. If it is not, a message will be returned asking the user to try a different tile. The client will then be prompted to input a float rating from 1.0 to 10.0 for the title they just inputted. Then, the client is asked if they would like to restart the cycle and input another title by answering 'yes.'

If they are done inputting titles, the client should input any value other than 'yes,' at which point they will be prompted to enter how many suggestions they would like to be returned. Using this input, a dictionary of anime titles to ratings is created, and new edges are added to the graph between users and the inputted animes, with the weights being the rating the client has inputted.

2) Then, we run Louvain's community detection algorithm on this graph in the graph.py file. As explained earlier in this section, this method clusters similar nodes within the network, and is taken from the NetworkX external library.

3) We then get the cluster that the program user is in using the partition variable.

4) In graph.py, the get_users_in_cluster function identifies the users that are a part of the client's community. From here, get_avg_weight_map finds the average score of each anime node in this cluster by each user in the cluster.

5) In main.py, the get_anime_suggestions takes these average ratings from within the client's cluster. The function then returns the title with the top average scores, based on how many suggestions the client wanted to be returned.

6) In conjunction with the returned list of titles, a bipartite graph is returned which shows the client's cluster. This sub cluster is created using the sub_cluster function. This sub cluster is then passed into the visualize_and_display function in visualize.py, which displays the sub cluster and the weights between the top anime suggested and the users in the cluster. Figure 4 is an example of what an outputted graph will look like - the blue nodes are anime titles, the yellow nodes are usernames, and the edges between the nodes are the previously discussed weights.

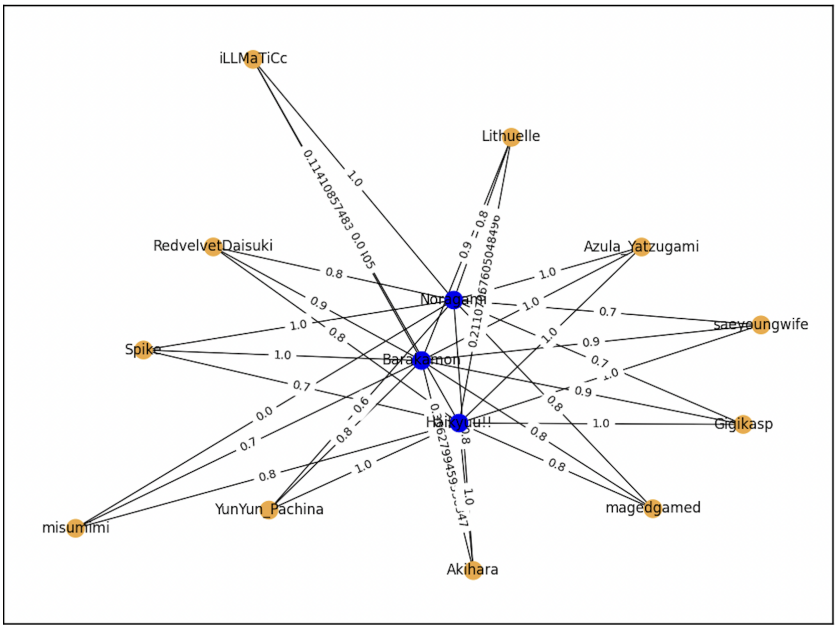

Figure 4: Sample Output - Bipartite Graph of the Anime Suggestion Simulator

Instructions

To use our algorithm, please follow these instructions:

1) Download the given zip file which contains both our datasets, and the original datasets in CSV file format. The original files are called UserAnimeList.csv and anime_kaggle.csv in this folder. Our files, which are smaller, modified versions of those files, are called animes.csv, genres.csv, and users.csv. Please ensure that these files are in the same folder as our py files, which include the code for our algorithm.

1 b) If you want to generate our CSV files using the original datasets, please follow these instructions:

i) Follow this link to the first dataset[1] which includes information about user's and their anime ratings: `https://www.kaggle.com/datasets/azathoth42/myanimelist?select=UserAnimeList.csv`

ii) From this webpage, download the UserAnimeList.csv file. Save this file's name as UserAnimeList.csv in the data folder in order to use the data extraction function in a later step.

iii) Follow this link to the second dataset[2] which includes information about animes on My Anime List and their corresponding genres: `https://www.kaggle.com/datasets/marlesson/myanimelist-dataset-animes-profiles-reviews`

iv) From this webpage, download the anime.csv file. Save this file's name as anime_kaggle.csv in the data folder in order to use the data extraction function in a later step.

v) In the main block, run the extract_data function using the two CSV files as the anime_kaggle_file and user_kaggle_file parameters. This function will extract the data from these files and make them into the three CSV files that we refer to in our algorithm: users.csv, animes.csv, and genres.csv.

2) Install the libraries listed under the requirements.txt file.

3) Open the main.py file and run it in the console. When prompted, input an anime title, and when prompted again, your rating (a float value) of the anime from 1.0 to 10.0. The anime title must be included in our datasets—if it is not, you will be asked to try inputting a new title. To add more inputs, answer "yes" when the program asks to continue. Your inputs will be stored in a dictionary.

4) When you are done inputting titles, input any word other than 'yes'. After a few moments, you will be prompted to input the number of suggestions you would like to be outputted. The number should be at least 1, and should not be a float.

For convenience, you may choose to use our pre-built anime preferences instead. To do this, uncomment one of the provided user_preferences dictionaries in the main block of main.py.

Here is an example of a user preferences dictionary:

{'Hunter x Hunter (2011)': 10.0, 'Yagate Kimi ni Naru': 8.2, 'Psycho-Pass': 8.1, 'Sword Art Online': 3.5, 'No Game No Life': 2.5, 'Black Clover': 5, 'Gin-

tama': 8.5, 'Kimetsu no Yaiba': 6.2, 'Ansatsu Kyoushitsu': 7.8, 'Angel Beats!': 8.0}

3 b) After following the above step, a graph that illustrates your results should automatically be opened up in a new window. However, if you want to create a graph from scratch following the methods that we used, please follow these instructions:

> We have provided a pickle file for our generated graph (program_graph.pkl) for your convenience. However, if you would like to generate the graph yourself, go to main.py and follow the instructions as outlined in the main block.

Changes to Project Plan

One change we made to our project is that our rating scale was initially from 0.0 to 1.0 (representing a score of 0 out of 10) instead of 1.0 to 10.0. This was changed because we realized after looking at our datasets that My Anime List's rating system is from 1 to 10. We assumed their system began at 0 because some users had 0 listed as their rating for certain animes in the dataset. However, this is because My Anime List uses a default value of 0 when users have chosen to leave a show unrated. Therefore, when the client is inputting their rating of shows, we did not want to give them the option to input 0, because in our algorithm, that would represent an unrated show.

Additionally, we initially intended to create our own dataset for user ratings and reviews by scraping the information from the My Anime List website. However, this would not have been ethical, as My Anime List forbids independent web-scraping on their servers. Instead, we found a dataset which included the information from My Anime List that we initially wanted to scrape.

Furthermore, we were initially planning to use the NetworkX external library only to visualise the bipartite graphs and the clusters that result from its community detection algorithm. However, we decided to use this external library to create the graphs as well as visualize them in order to maximize efficiency, and to avoid using unnecessary classes and instances.

Discussion Section

The results of our algorithm reflect our initial goal — a client can use our code to input a list of animes they like, and they will receive a list of new animes that we think they would enjoy. However, there are certain factors we were not able to fully account for that could have improved the accuracy of our algorithm.

One limitation of our algorithm is how it deals with shows that are included in

our dataset, but do not have any user ratings. Although it is a rare case, there are certain shows in the dataset that have a default score of 0 for every user that has seen it. This means that no user has rated this anime, and thus, there is not enough information to create an accurate cluster for the client in order to recommend certain suggestions. Based on our observations, this does not occur when the anime belongs to a popular combination of genres.

To combat this, we return a message to the client if they input an anime that falls into this rare case, which says that there is "not enough data to give recommendations; no other users in cluster." This should occur for niche animes, or those which were released around the same time our datasets were created, and thus did not have people who had rated it yet.

Secondly, our matrix multiplication is intended to map users to every anime in our dataset, using information about how much other users liked a particular show. However, not every user has seen every anime. In order to complete the matrix multiplication, we needed a default value for these 'empty spaces' that arise when a user has not seen a particular anime. This situation is slightly different than the one above - if no users have seen a certain anime, *and* a user has not seen every anime that belongs to a certain combination of genres, we must output that there is not enough information for an accurate cluster.

However, both of these cases do not always occur. When a user has not seen every anime under a certain genre, this does not necessarily mean that the anime has not been rated by every user in the dataset. We account for this in-between case with a default value.

We chose to use 0.5 as this default value, as it is in the middle of the user rating scale of 0.1 - 1.0, and therefore does not skew the results particularly negatively or positively. One could argue that this is a problem, because we are assuming that if these users had seen the animes, they would have rated them 5 out of 10, and we do not know this for certain. Despite this assumption, we still feel that this was the best choice of value, given that we required a default value that does not skew the results of our matrix multiplication.

Additionally, the set of animes that we included in our algorithm is limited. We were unable to include every anime title that has been produced inside of our dataset, which means our algorithm will work best for clients who have seen animes that we have accounted for in the implementation. Similarly, our matrix multiplication takes in a maximum of 100 users for the sake of running time and efficiency.

However, since our data is taken from one of the most popular anime rating websites in the world, we feel that our data is reflective of many anime-watchers' viewing history, and thus our algorithm should be applicable to a large audience. There is potential for our algorithm to be extended so that it includes a

much larger data set and therefore is helpful to more anime-watchers.

Because we were not able to use web-scraping to obtain data for this project and instead resorted to an existing dataset, our algorithm does not include up-to-date information on ratings and new titles. If a new show were to come out that clients would like to use as part of their input, our algorithm would need to be updated to include this title. Or, if the quality of a show suddenly declined and its ratings went down, our algorithm would not immediately reflect this and may continue to recommend that show to users. We feel that these cases do not occur frequently enough that they impede the accuracy and usage of our algorithm. This program could be extended so that its dataset is updated periodically to include more up-to-date information.

We feel that there is the potential for the above factors to be accounted for in extensions or future versions of our algorithm. Given the resources that were available, we feel that our algorithm is accurate and fulfills our initial goal of aiding anime-watchers in finding new titles that will suit their interests. In an age where there is an overwhelming amount of information and opinions available at any given time, our simple but useful algorithm provides a helping hand by recommending new titles which account for modern preferences, without requiring clients to wade through thousands of ratings and reviews themselves.

Notes

[1] "MyAnimeList Dataset." *Kaggle*, https://www.kaggle.com/datasets/azathoth42/myanimelist?select=UserAnimeList.csv.

[2] "Anime Dataset with Reviews - MyAnimeList" *Kaggle,* https://www.kaggle.com/datasets/marlesson/myanimelist-dataset-animes-profiles-reviews.

Works Cited

"Anime Dataset with Reviews - MyAnimeList." *Kaggle,* https://www.kaggle.com/datasets/marlesson/myanimelist-dataset-animes-profiles-reviews.

"Community detection: the Louvain algorithm." *atlas.brussels*, https://atlas.brussels/community-detection-louvain-method/.

"Content Based Filtering Vs Collaborative Filtering." *Research Gate*, 13 Feb. 2018, www.researchgate.net/figure/Content-based-filtering-vs-Collaborative-filtering-Source_fig5_323726564.

Kurama, Vihar. "What Is Collaborative Filtering: A Simple Introduction." *Builtin*, 29 Aug, 2022,

Lancichinetti, Andrea, and Santo Fortunato. "Community Detection Algorithms: A Comparative Analysis." *Physical Review E, vol. 80, no. 5*, 2009, doi:10.1103/physreve.80.056117.

"Matrix multiplication diagram." *Research Gate*, 13 Oct. 2019, https://www.researchgate.net/figure/Matrix-multiplication-diagram_fig5_299679931.

"MyAnimeList Dataset." *Kaggle*, https://www.kaggle.com/datasets/azathoth42/myanimelist?select=UserAnimeList.csv.

"Scrapy 2.8 Documentation." *Scrapy.org*, 2008, docs.scrapy.org/en/latest.

Shetty, Badreesh. "An In-Depth Guide to How Recommender Systems Work." *Builtin*, 24 July, 2019, https://builtin.com/data-science/recommender-systems.

Vatsal P. "Louvain's Algorithm for Community Detection in Python." *Medium, Towards Data Science*, 21 March, 2022, https://towardsdatascience.com/louvains-algorithm-for-community-detection-in-python-95ff7f675306.

Vatsal, P. "Recommendation Systems Explained." *Medium, Towards Data Science*, 20 May 2022, https://towardsdatascience.com/recommendation-systems-explained-a42fc60591ed.