Spring 2019

Building Recommendation Systems

Orion Davis
ord4@zips.uakron.edu

Building Recommendation Systems

Orion Davis

April 23, 2019

1 Introduction

Recommendation systems are pieces of software that suggest new items to a user.[1] There are many moving parts to these systems including data, the actual recommendation model, processing data and finally displaying data. This project explores the role each part plays in the overall system and how to develop a recommendation system for beer from scratch.

To answer these questions an Android application was built to communicate to the built recommendation system through a series of application programming interface (API) calls. Data for the application was kept in a NoSQL graph database on the same server that ran the back end recommendation service. The Android application asked the server for specific information and would then render the returned information to the user.

This paper begins with outlining different types of recommendation systems and some details on how they function. Then, the specific process for calculating user similarity is described in detail to help the reader understand how the beer recommendations in this system are being calculated. Next, the paper describes the technical implementation of the recommendation system and its different technologies. The paper then concludes with a summary and the future work considerations of this project.

2 Types of Recommendation Systems

The International Research Journal of Advanced Engineering and Science conducted a survey on categorizing different types of recommendation systems. Their findings list seven unique types of recommendation systems falling into a few different categories including content based and collaborative filtering, among a few others.[2] These are the two main classifications of recommendation systems that this project will explore.

2.1 Content-Based

A content-based recommendation system is built on the idea that items are described and labeled with specific keywords. Users then create profiles which indicate what kind of items they like and do not like based on the different labels used by the system. Using a user's preferences the system can then recommend new items that share similar properties to the items that fit the

user's known likes.[3] As it can be seen these systems require some knowledge of the data that is being recommended and how to describe different items. This would require acquiring a data set and then processing the data and attaching labels to describe the items.

A disadvantage to implementing this type system is labeling all the beer with different descriptors. This work would have required a deeper understanding of beer and how to describe it, rather than the math behind calculating recommendations. Because of this distraction from the math, a content-based recommendation system was not implemented. Instead, options that don't need knowledge of the data they are recommending were pursued.

2.2 Collaborative Filtering

Fortunately collaborative filtering traditionally does not require knowledge of the data which is being recommended. Collaborative filtering works on examining a user's past behaviors and predicting their future behaviors based on users similar to them.[4] The system just needs to know what items a user has rated and find other uses that are very similar to this user. In finding similar users the system can then recommend items that they like which in turn the original user may find they like too. The complexity in collaborative filtering comes in determining which users are similar to which.

There are many different mathematical models to perform user to user similarity calculations. Most models are some system of finding a mathematical distance between users based on their rating patterns on common items. Choosing from these different models is where a large portion of time was spent researching and finding the best model for this system.

Overall, collaborative filtering was chosen as the recommendation system because it allowed me to focus on the math behind making recommendations and predicting user behavior purely based on numbers. This simplified the approach and did not require extensive knowledge on the classification of different beers.

3 Calculating User Similarity

There are many different methods for calculating similarity of users solely based on the rating that they give different items. Examples include k-

nearest neighbor, cosine similarity, Pearson correlation and Euclidean distance. These were some of the more heavily considered algorithms as they operate on items that two users have both rated rather than similarity of the items themselves. In the end, I chose to implement the Pearson Correlation for calculating the similarity between different users of the recommendation system. I chose Pearson correlation because it was very intuitive to calculate beer recommendations using only the rating data provided by the users.

3.1 Pearson Correlation

Specifically, this system implements the sample Pearson correlation coefficient. In calculating a predicted rating for an item for a user the system finds all users who have rated that particular item. Next, the sample Pearson correlation coefficient is calculated between the asking user and all other users who rated the specific item as shown by formula 1.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{1}$$

Here, r is the similarity coefficient between user x and y. A user's item score is denoted by either x_i or y_i across the set of items scored by both users n . Each user has their average item score denoted by $\bar{x}$ and $\bar{y}$ respectively. Including a user's average score helps curb the effect of extreme ratings between users. For example, for one user a great beer may be rated a 5, but for others they may only rate great beers a 4. In a way, this creates a pattern for how a user rates and helps keep from extreme ratings on either side of the scale from vastly affecting the similarity between two users.

With the Pearson Correlation coefficient for two users calculated you can then calculate, using formula 2, a predicted rating for a specific item, which for this system happens to be beer. [5]

$$p_{u,i} = \bar{r_u} + \frac{\sum_{v \in nn(u)} sim(u,v)(r_{v,i} - \bar{r_v})}{\sum_{v \in nn(u)} \mid sim(u,v) \mid} \tag{2}$$

This calculates the predicted rating for user u for item i. In the equation, v $\in nn(u)$ denotes the set of k-nearest neighbors to user u, or, the k-most similar neighbors to u. The predicted rating also takes into account each user's average rating, similar to when calculating the actual coefficient to offset the effect of extreme scores.

4 Implementation

With this new understanding of the recommendation model that was selected the actual application implementation can be started. There are several components to the application which are listed and described below.

- Database
- REST API
- Android application

4.1 Database

Due to the nature of recommendation systems being a series of relationships between users and beers a graph database was chosen. Neo4j is a NoSQL graph database which provides intuitive methods for creating nodes and edges in the graph. Each user and beer is represented by a node in the graph and when a rating relationship between the two is created an edge is created in the graph. An example of these relationships can be seen in figure 1.

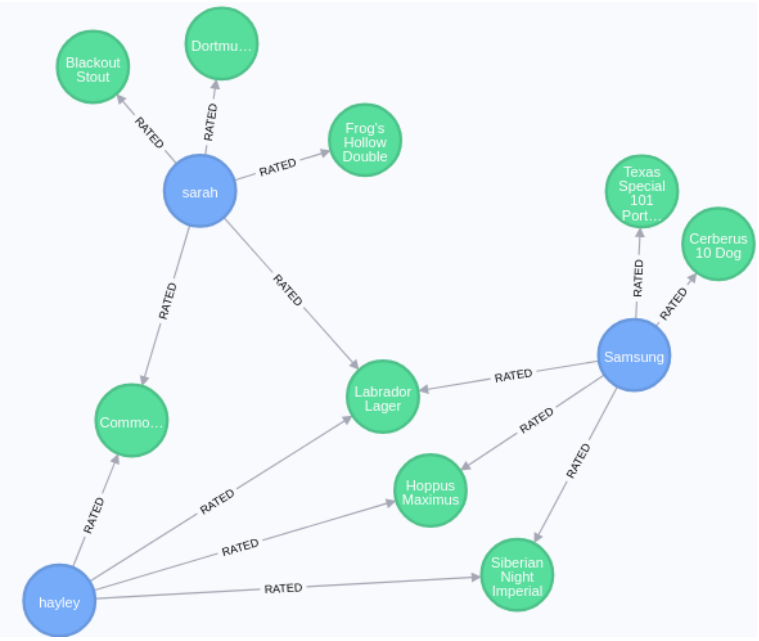

Figure 1: Sample node and relationships

Blue nodes represent different users while the green nodes represent different beers. You can see that there are beers that have been rated by multiple users by looking at the *RATED* edges in the graph. Within these edges the value of the rating is held and is where the rating data is pulled from when calculating beer recommendations. Consider the example where the user *hayley* wants to try the *Cerberus 10 Dog* beer but is unsure if she will like it. Through the three beers rated by both *hayley* and *Samsung* their similarity is calculated with Pearson's correlation. Because *Samsung* has rated the beer that she is curious about using formula 2 the system can report a predicted rating to *hayley* and this can help her decide whether or not she wants to try the beer.

For populating the database, a data set from OpenBeerDB was used. This data set had approximately 4,400 beers of varying styles and from many different breweries.[6] This allowed users to start rating beers immediately rather than requiring all the beers be entered manually.

4.2 REST API

Using Python Flask, a web micro-framework, a REST API was created to expose all functionality for the recommendation system. This API allowed users from the Android application to make requests to the server for specific information. Example calls to the API include creating a new user, rating a beer, getting a predicted rating and creating a new beer. To handle all the requests different endpoints where defined to perform the different POST, PUT, GET and DELETE requests. All request bodies for requests made to the API were in JavaScript Object Notation (JSON) as this allowed a standardized way of communicating between the client and server. This API lives on a server at a location that can be reached by any instance of the Android application trying to make requests.

4.3 Android Application

An Android application serves as the interface in which user's interact. The application itself is written in Kotlin and the pages the users see were created through XML. Through using Volley, an included HTTP library, the application can post its requests to the server and receive back the data asked for in the request. When a user first begins the application they are asked to

either sign in or create a new account. Once into the system the app displays a list of beers to the user holding information such as the beer's name, brewery and style. Users also have the option of searching for beers based on either their name, their brewery or their style. With this request made the server will return all beers that match the criteria entered by the user. From the list of beers the user can click on any of the beers to see any additional information about the node there may be in the database along with either the user's rating for the beer, or the predicted. In this view, users can rate or update their rating which will then be reflected in the database through the appropriate API call.

5 Challenges

Through implementing a recommendation system some issues arose and needed to be addressed. One such issue is the efficiency and speed of the system. Calculating the Pearson Correlation can take a lot of time as the number of users and size of shared ratings between users increases. A balance must be struck in maintaining accurate recommendations without hogging the systems resources to be calculating the similarity between users each time a beer is loaded. To solve this problem user to user similarity is calculated at the beginning of their app session. The back end then will store up to twenty of the most similar users to the one that just logged in. These similarities will then be used in calculating the predicted rating of an asked beer. This allows the system to keep similar users up to date to aid in better recommendations without sacrificing efficiency and speed.

Another prevalent challenge to this implementation is lack of users and its effect on ability to make recommendations and their quality. However, due to the nature of this project wanting to focus on the parts of a recommendation system and what goes into them this challenge can be pushed aside.

6 Conclusions & Future Work

This project helped tie together many different aspects of the Computer Science curriculum here at the University of Akron from database management to creating client-server applications. Additionally skills for developing full stack applications were gained and practiced through the creation of the back

end recommendation system and a user facing Android application. This recommendation system utilizes Pearson correlation due to the intuitive nature for calculating beer recommendations based off a user's previous behaviors.

Some planned work for this project in the future includes implementing a safety net, or even a hybrid system, for aiding in providing recommendations to new users. Additionally, some data analysis on user's specific likes and dislikes is planned, as this can also aid in gathering recommendations for new users. Improving the user experience for the Android application is also planned.

Overall, this project challenged my programming abilities to create a beer recommendation system and helped bring together my entire education here at the University of Akron. It was a unique project of great interest to me that will continue to develop past graduation.

References

[1] Ricci F., Rokach L., Shapira B. (2011). *Introduction to Recommender Systems Handbook*. Retrieved from http://www.inf.unibz.it/ ricci/papers/intro-rec-sys-handbook.pdf

[2] Gigimol, S., Sincy, J. (2016). *A Survey on Different Types of Recommendation Systems*. International Research Journal of Advanced Engineering and Science. Volume 1, Issue 4, pp. 111-113.

[3] Billsus D., Pizzani M. (2007). *Content-Based Recommendation Systems*. In: Brusilovsky P., Kobsa A., Nejdl W. (eds) The Adaptive Web. Lecture Notes in Computer Science, vol 4321. Springer, Berlin, Heidelberg

[4] Melville P., Sindhwani V. (DATE). *Encyclopedia of Machine Learning*. Retrieved from http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf

[5] Ruozzi N. (2015) *Collaborative Filtering*. Retrieved from https://www.utdallas.edu/ nrr150130/cs6375/2015fa/lects/Lecture_23_CF.pdf

[6] *OpenBeerDB*. Retrieved from https://www.openbeerdb.com