Stefan Urosevic - 13su12 - 10146785
CISC 467 - Fuzzy Logic - Implementation

<u>Using Fuzzy Logic to Generate Movie Suggestions Using Ratings From Other Users</u>

I decided to use fuzzy logic to generate movie suggestions for users from other users' data. I have seen systems like this in IMDB, Netflix, and other movie database website. I wanted to see how well fuzzy logic would work in these situations. The aim of my system is to gather movie rating data from the user and to create a fuzzy set with it. It then uses that fuzzy set to compare how similar the ratings of other users' are. Using this similarity comparison, it suggests movies that it thinks the user would like based on the ratings of other users.

Implementation Details

readCSV - This function reads csv data and returns a list of that data. It was used to read in the movie and ratings data that I used to compare similarity.

createMovieDict - This function takes a list of movies and corresponding IDs and creates a dictionary mapping a movie to the correct id. This was used as a list of movies as well as a way to convert movie ids into movie titles

parseUsers - This function was used to read the rating csv file. It used the list returned from readCSV and outputted the data in the form of a dictionary. They keys were the ids of the users and each user had a corresponding dictionary. This nested dictionary contained the rating data for that user. The keys for this ditionary were the movie titles and the values were the ratings. This function converted the ratings from stars into values in the range [0 .. 1]. These were the fuzzy sets of ratings that the program compares against.

getInput - This function was in charge of getting input from the user and outputting a dictionary where the keys were movie titles and the values were ratings in the range [0 .. 1]. This was the users' fuzzy set for ratings.

Unit_vector, angle_between - These 2 functions were taken from stack overflow (https://stackoverflow.com/questions/2827393/angles-between-two-n-dimensional-vectors-in-python/13849249#13849249). They are basic vector math functions. They take a 2 vectors and return the angle between them. This was used to compare the similarity of 2 users. The vectors in question are the fuzzy sets of the users being compared.

Sim - This function compares the similarity of 2 users' ratings. It checks if the users have any movies in common and then uses the angle_between function to get the angle between the fuzzy sets as vectors. It then applies COS to this angle to find a similarity value in the range [0 .. 1]. If the users have no movies in common, the similarity is 0. If they have every movie in common, the similarity is also 0. This is because 2 users that have every movie in common have no possible movies to suggest and so they are useless for suggesting movies. These cases are assigned a similarity value of 0 so that they are not considered when suggesting movies.

compareRatings - This function loops through the database of user ratings and compares each user in the database to the user that just entered their ratings. It then compiles a list of similarity values and sorts it with the users with the highest similarity being first in the list. It then returns the top 25 users in the resulting list.

getRatedMovies - This function only takes all the user ratings and returns the list of movies that a specific user has watched

suggestMovies - This function takes a list of users and collects a list of all the movies that list of users has collectively watched. It then randomises the order of that list, removes duplicates, and chooses the first 15 movies. These are the movies that will be suggested to the user looking for suggestions.

main - This function creates the movie dictionary, then creates the rating database, then gets input from the user. It gets different input depending on whether the user wants to enter their own ratings or be assigned a random users' movie ratings. These are the ratings that will be used to suggest movies. The latter option is included for ease of use of this program. The highest similarity matches to users are found and then the movies that those users have watched are suggested to the user using the program.

Program Flow and Thoughts

Here I will go through the implementation of this movie suggestion algorithm and explain the choices that were made that affected the design of the program.

Fuzzy Sets - Fuzzy sets were represented as dictionaries where the values of the keys were the values of the fuzzy sets. This made comparing 2 fuzzy sets easy as python has plenty of functionality that makes dictionaries easy. It also allows me to keep an order to the fuzzy values without needing to keep track of the index they were at. The keys, the movie names, were used to keep track of which fuzzy value was where.

Comparing similarity - I decided to compare a relative similarity rather than an absolute similarity. By that, I mean that I compare the user ratings relative to the other ratings that the user made rather than purely by the rating they gave to a movie. This allows for situations where one user may not like movies very much but likes the same type of movies as another user. This user would still be able to provide movie suggestions because their relative rating of movies is the same as the relative rating of the second user. User A might rate more harshly than B but they might still like the same type of movies.

Comparing relative ratings means this situation is still useful. I also made the choice of not comparing users that did not have anything in common and users that had everything in common. The former is pretty obvious, I do not have data to compare users that have nothing in common. The latter was a design choice due to the outcome of it. If 2 users have every movie in common (the only situation that actually matters is that the database user, user A, has seen every movie that the user looking for suggestions, user B, has seen), then user A will not be able to suggest any new movies to user B, even if they are a perfect match. Since this system is made to suggest movies, situations like these are useless. I do not want to suggest a movie that a user has already seen. To get around this, if the users have both seen the same movies, I give them a similarity rating of 0. This means that they will not be used to generate suggestions once all of the other database users have been looked at.

Suggesting movies - They way I suggest movies is I get the top 25 users that have the most similar ratings as the user in question, and make a list of all the movies that the 25 users have collectively rated. I then remove any duplicates and remove any movies that the user in question has already rated. Once I have this list, I shuffle it and choose the first 15 movies to suggest. The reason I do this is because there may be many users with the same highest similarity so I take all of those into account. I then shuffle it so that the user may ask for more suggestions and they would get different movie suggestions. Shuffling and choosing the top 15 is an arbitrary action and I only do it because It is a slightly neater way of displaying suggested movies. Otherwise, there would be too many movies to conveniently display.

Possible Issues

An issue that can come from this system is that I don't check if the user from the database likes the movie I will suggest. The reason I do not do this is that some users may rate movies more harshly than other users and so I would need to check the rating compared to the rest of their ratings to see if they enjoyed the movie or not. This is something that can be done quite easily and should be done if this system is moved to a production environment. I decided to exclude it as it did not display the concept of using fuzzy logic in a movie suggestion system any better than not including it.

Another issue comes from using relative comparison rather than absolute comparison. Suppose I say "Toy Story" is 5 stars and "Jumanji" is 2.5 stars. Suppose user B says "Toy Story" is 2 stars, "Jumanji" is 1 star, and "Some Like it Hot" is 5 stars. My system only compares movies that both have seen to find similarity. Since I use relative similarity, user A and user B would have a very high similarity rating because the relative rating of "Toy Story" and "Jumanji" is the same. That would then cause "Some Like it Hot" to be suggested to user A. This may not be a good recommendation since user B did not like the first 2 movies very much compared to "Some Like it Hot", however, the relative comparison thinks it would be a good match. This problem can be avoided in the future by moving to an absolute comparison method instead of relative. This would mean that ratings themselves are compared for similarity rather than relative ratings being compared. I chose to use relative ratings to prevent situations where user A has a much harsher rating system compared to user B. The appropriate rating system would depend on what type of production environment this will be used in.