CS51 Final Project: Final Spec

TOONZ (final name)

Rachael Smith ~ Anna Zhong ~ Alex Lehman

Brief Overview:

We would like to build a program which can recommend new music to users based on their previous preferences. In order to do this, we will first implement a neighborhood-based collaborative filter. The neighborhood CF will create a database of item-item similarities; it will enable us to answer simple questions like "give me more songs like this one". We chose to use an item based CF, as realistically people do not generally rate a lot of things so a user-user based CF would have a really sparse data set. Next, we want to write an algorithem which will use the neighborhood CF to predict the rating a user will give to a particular song. To do this, we will use the CF to calculate the songs most similar to a given song, from the set of songs which the user has already rated. Then, we will incorporate the ratings of those 'similar songs' to predict the rating the user will give that particular song.

Feature List & Technical Specifications

CORE FEATURES[i]

1) An item-item Collaborative filter
    a. This involves calculating similarities between songs based on a database of user ratings. This will only work if multiple users have rated the songs, otherwise there is no basis for the rating.
    b. Then we will create a database of these similarities. The database will contain every possible pair of songs and the value represent their degree of similarity.
2) A function which returns "songs similar to song X"
    a. This involves implanting a "k-nearest- neighbors" function
    b. This function will provide top "global" ratings, that is, it will not be specific to the user
3) An algorithm to predict user ratings of a song
    a. First, we will need to find the "k-nearest-neighbors" to a song out of the set of songs the user has already listened to.
    b. Then we will use the ratings of the near neighbors to predict the user's rating by the algorithm given in section 3.2 of the aforementioned paper.

HOPEFUL FEATURE[ii]

In order to make a maximally accurate algorithm, it would be best to take a combined approach to our CF. Ideally, we'd like to also implement a latent-factor based recommender. This would be a user-user style recommender. I would compute similarities between *users* rather than items, operating under the idea that similar users will rate similar restaurants similarly. In making our recommendations to users we would combine the results of the item-item CF with the results from the user-user CF.

EXTRA FEATURES

1) Embedding a music player into the user interface
2) Taking into account whether or not the user tends to give higher or lower ratings as compared to the average ratings when building our database of similarities
3) Implement map-reduce paradigm to allow for scalability of our algorithm.
4) Designing a user interface for our algorithm

PLAN FOR MODULARIZATION/CODE OUTLINE

Choosing a relevant API and importing the API into a useful data-frame
This has been completed! (see "Progress" below)

Building a Song Similarities Database[iii]:
Our song similarities database is instantiated by a python class. Outlined briefly the class should look something like this[1]:

class DB
    def **init** *the constructor method. Initializes the class.*
    def **populate** *populates the database with song similarity information derived from helper functions*
    def **get** *method which allows for retrieval of information from database*

Our helper functions which allow us to populate the database will have the following signatures and functionality:

def get_songpair_users(song1_id, song2_id, df)
    inputs: song1_id and song2_id represents 2 particular songs in the dataframe, df represents the dataframe
    outputs: a list of (shared_user, song2_count, song1_count). That is, it is a list of the shared listeners of the song and that user's play counts.

def sim_funct(song1_id, song2_id, user_list)
    inputs: pair of songs (song1_id, song2_id) and the list of shared listeners of the pair (user_list)

[1] Adapted from Pfister, Hanspeter. " HW4: Do we really need Chocolate Recommendations?." CS 109, < http://nbviewer.ipython.org/github/cs109/content/blob/master/HW4.ipynb> Oct. 2013. Web. 10 Apr. 2014.

- for each particular shared listener of (A,B) : takes the difference between that user's number of listens for that particular song and the user's average listens (to adjust for individual differences in listening quantity) like so:
diffA = song_listensA – average_song_listens
diffB = song_listensB – average_song_listens
- Then a correlation function is run on diffA and diffB to calculate the similarity of the two songs

Output: a number representing the degree of similarity of songs

In order to make the similarities database we will populate the database by running sim_funct.

Giving Global Recommendations[iv]

This section of the project will interpret the data contained in the database we created in the previous part. It will allow for giving 'global' song recommendations, that is, it will recommend songs based on the global song similarities, as calculated in forming the database in the previous part. The recommendations will not be tailored to the particular user.

K_nearest_neighbors(song_id, song_set, db, k)

Inputs: particular song we want to find the most similar songs to, a set of songs to search through, our similarities database, and the number of similar songs we wish to output
- Searches through the database for the songs closest to the given song using the get method

Outputs: the *k* most similar songs

Top_user_recs(user_id, db, df, k, n)

-*Inputs:* an individual user id, the similarities database, the entire dataframe, k and n, two integers
-searches the dataframe for the n most listened to songs of that user
-runs k_nearest_neighbors on the most listened to songs of the user to find the k most listened to songs
- filters out songs user has already listened to/duplicates

Outputs a ranked list of songs to recommend to the user

Predicting User 'Ratings'[v]

This portion of the project is where we implement the collaborative filtering algorithm. We haven't mapped out the pseudo code for this section of our the project, but we know it will rely heavily on the previous two sections, as the collaborative filter will draw upon the similarities database. We spent much of our time preparing for this section of the project researching different collaborative filtering algorithms. Our algorithm will be adapted from the approach taken by Badrul Sarwar in his paper, "Item-Based Collaborative Filtering Recommendation Algorithms" and from the approach outlined in course materials provided by CS109 course materials.

As our API does not provide us with information regarding user ratings, only song counts we are going to adapt the CF algorithm to predict the number of times a user would listen to a song, rather than to predict the rating a user would give.

Our model will first predict a baseline value for the user's number of listens predicted for a given song. A baseline estimate in an item-item collaborative filtering model can be given by:

$$B_{us} = m + (m_u - m) + (m_s - m)$$

Where m is the average number of listens of all the songs over all the users in the dataset, m_u is the average number of listens for the user across all songs, and m_s represents the average number of listens

the particular song has across all users. The second and third terms of the expression are meant to count for user biases and item-specific biases in the data.
Then we will employ the following collaborative-filter model:

$$R_{ut} = B_{ut} + \frac{\sum_{i\epsilon N^k(t)} s_{jt}\,(R_{uj} - B_{uj})}{\sum_{i\epsilon N^k(t)} s_{jt}}$$

Here, R_{ut} is the predicted number of listens of song t for user u. $N^k(t)$ represents a function which gets the k nearest neighbors of song t among the songs user u has listened to. s_{jt} represents the similarity between two songs $j$and t (as found in our similarities database). $R_{uj} - B_{uj}$is the difference between the number of times u actually listened to song j and the baseline prediction of how many times $u$would listen to j.
We will need to implement helper functions to calculate $N^k(t)$, B_{us}, and likely a helper function to extract relevant values of s_{jt} from our similarities database in order to build R_{ut}.

PROGRESS REPORT

DATA
We selected an API to use as our source of data for this project. We will be using the EchoNest song API. We chose this particular API because (1) it is very large, and the larger the dataset, the more accurate the results will be. And (2) there is a “Taste-Profile Subset” of this API which contains information about how much users like a particular song in the form of play count. We had difficulty finding an API with rating information, so we decided to adapt our rating system to be based on play counts (as reflected in changes to our PLAN FOR MODULARIZATION/CODE OUTLINE)
After reading the API into a pandas dataframe, we manipulated the dataframe to simplify things down the road. We calculate how many unique songs each user has listened to and how many unique users have listened to each song. We added columns containing this data to our dataframe. This data is important to understand the user-user, song-song, and user-song relationships in our data set in order for us to build our recommender system.
Similarities Database
We have fully outlined all the functions we will need to build our database class and have begun implementing them.
Item-Item Collaborative Filter Algorithm
We have researched and selected an algorithm to implement.

SOURCES

Wikipedia contributors. "Collaborative filtering." *Wikipedia, The Free Encyclopedia*. Wikipedia, The Free Encyclopedia, 25 Apr. 2014. Web. 29 Apr. 2014.

Sarwar, Badrul. *Item-Based Collaborative Filtering Recommendation Algorithms*. Tech. N.p.: n.p., n.d. GroupLens Research Group/Army HPC Research Center Department of Computer Science and Engineering University of Minnesota. Web. 12 Apr. 2014.

Pfister, Hanspeter. " *HW4: Do we really need Chocolate Recommendations?*." *CS 109*, < http://nbviewer.ipython.org/github/cs109/content/blob/master/HW4.ipynb> Oct. 2013. Web. 10 Apr. 2014

[i] All our Core Features were implemented as described in “Modularization Plan”

[ii] Unfortunately, we did not end up implementing this feature, nor any of the following “extra” features.

[iii] We ended up implementing the DBclass as described.

However the helper functions used to populate our database ended up being implemented slightly differently. Rather than creating a list of song pairs and shared listeners, we created a function which just retrieves the sub-dataframe of users who have listened to a particular song (get_song_listeners) and we created a function which returns the sub-data frame of play counts of a particular song over a set of users (get_song_counts).

We created a new function song_sim which integrates all of these helper functions into a single function. Song_sim calls get_song_listeners on two songs and intersects the results, to get the set of shared listeners (common). Then get_song_counts is called on each song over the common list. Our sim_funct then operates on the outputs of our calls to get_song_counts. The result of sim_funct is returned by song_sim. Song_sim is used to populate our DB class, rather than sim_funct.

[iv] Implemented As Described

[v] Implemented As Described