CS51 Final Project: Draft Spec

TUNEZ (tentative name)

RACHAEL SMITH ~ rachaelsmith@college.harvard.edu
ANNA ZHONG ~ azhong@college.harvard.edu
ALEX LEHMAN ~ ealehman91@gmail.com

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 algorithm 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.

Our primary goal for the project is to finish. Ideally, this will involve a completed recommender system which can recommend songs with a (reasonably) low margin of error.

Feature List & Technical Specifications

CORE FEATURES

The following Core Features will be implemented as described in Badrul Sarwar's paper "Item-Based Collaborative Filtering Recommendation" (http://files.grouplens.org/papers/www10_sarwar.pdf)

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

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. The second CF would be implemented as described in http://dl.acm.org/citation.cfm?id=963774

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

Choosing a relevant API and importing the API into a useful data-frame

We will need to do some research to find an API that contains song data along with user rating data. We will need to import the API into some kind of useful data structure. We intend to do this using pandas data frames.

Building a Song Similarities Database:

Item-item collaborative filter algorithms rely on using similarities between items to provide user recommendations. So for the first step in our project, using a python class we want to create a database of similarities between all the songs in our API. So, first we will need to create a database class, then we will need to populate the database with relevant information. We anticipate that the main functions required to implement this part will be as follows:

common_listeners

-goes through each song in the database
- for each pair of songs, calculates the number of shared listeners
-outputs a new dataset containing the pairs of songs and the shared listeners

song_sim

-takes a song, song pair (A,B) and the list of shared listeners of the pair

- for each particular shared listener of (A,B) : takes the difference between that user's rating for that particular song and the user's average rating (to adjust for individual differences in rating scales) like so:

diffA = song_ratingA – average_song_rate

diffB = song_ratingB – average_song_rate

- Then a correlation function is run on diffA and diffB to calculate the similarity of the two songs
- Outputs a number representing the degree of similarity of songs

Similarity_database

-iterates through the entire song database

- runs song_sim on each possible pair of songs if common_listeners is higher than 0

-outputs a database containing the pair of songs, their similarity, and their common listeners

We will need to research what kinds of similarity functions will be appropriate for our purposes and how to create a useful database class in python.

Giving Global Recommendations

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. This will provide interesting information about relationships between songs in our database, and it will also be a way to test how well our similarities database works.

K_nearest_neighbors

Given a single song, and the similarity database, outputs the *k* most similar songs

Top_user_recs

- Takes the n most liked songs for a user
- runs k_nearest_neighbors on those songs
- filters out songs user has already listened to/duplicates
- outputs a ranked list of songs to recommend to user.

Predicting User Ratings

This section is where we implement a collaborative filtering algorithm to tailor song recommendations to a particular user. We will need to do more research into how to implement a user-user CF, however we anticipate that implementing it will break down into the following three steps:

K_nearest_users_songs

- Analogous to k_nearest_neighbors
- given a song and a particular user, outputs the k-nearest neighbors to the song from among the list of songs the user has listened to

Baseline_estimate

- Given a song a user has never listened to, gives a baseline estimate for how much the user will like the song
- Let A = (average # rating of song A over all users)
  Let A_u = average # rating over all the users songs
  Let A_s = average of users rating for song A
  Baseline estimate= $A + (A_u - A) + (A_s - A)$

Predict_rating

-given a song which a user has never listened to, predicts how much a user will like it

- Here the collaborative filtering function will be implemented. We imagine it will be somewhat like the one presented at http://files.grouplens.org/papers/www10_sarwar.pdf which adjusts the prediction for how much the user will like it from the baseline, given the k_nearest_users_songs

Next Steps

1) Research song APIs
2) Download iPython notebook, familiarize ourselves with relevant aspects of Python (including how to use pandas dataframes)
3) Begin implementing code for similarities database