Project09 -- KDD track 1

Chong Ding

**1. Knowing the data**

Knowing the data format is the start of work on this project. Spend some time to be familiar with the data can help build some right and efficient structure to manage the data for user and different items.

**2. Strategy**

|  |
| --- |
| typedef struct{  string id;  int score;  int ideal\_score;  float confidence;  }item;  typedef struct{  string id;  int numItems;  vector<item> items;  }user;  typedef struct{  string trackId;  string albumId;  string artistId;  vector<string> genres;  }track;  typedef struct{  string albumId;  string artistId;  vector<string> genres;  }album;  typedef struct{  album ialbum;  string iartist;  track itrack;  string igenre;  string type;  }itemType; |

Structures item, user, track, album, itemType are used to contain and manage the data. Itemtype is a special one used when trying to find the type of one item in the user’s rating pool. It could be genre, artist, track or album. So *itemType get\_item\_type(string id)* return a itemType structure containing the data of this item.

**Prediction score strategy:** For each user *i* in the validation data set, compare it to all the users in the training data set to find the user has the maximum similarity with it or sort the similarities between *i* and all the users in the training data. The score predicted is primarily based on the *i’*smost similar user. If the item in *i* can be found in the most similar user’s items pool as well, then give the same score and confidence is 1, if not, try to clarify the item’s type first, then trying to find the similar item (same type, similar attributes) in the most similar user’s items pool, score it with the most similar item’s score. If no similar item found, no score will be predicted, so the predicted score is negative value (-1), which is the initialized value. It can be done by compare it with the second most matching user in training dataset. In the code, I just compare the user with the most matching user in training data set to predict score. If the second most matching user doesn't help predict decent score, it can goes to the third most matching user, so on and so forth. But runtime will increase, so I didn't let my program do this right now.

There are some items’ score are still -1, which can be simply solved by trying to find similar item in the next most matching user’s items pool, update it if higher confidence can be reached. For lower matching user, the confidence is the similarity between items with same type multiplied by a coefficient 0.90^(k-level), k-level is the k-th matching user.

**Similarity routines:**

Same track or album: float similarity\_track\_album(itemType one, itemType two)

For track, it’s based on whether they have same artist, same album or same genre. Same artist gives 5 credits, same album gives 3 credits, same genre gives 5 credits.

For album, it’s based on whether they have same artist, or same genre. Same artist gives 10 credits, same genre gives 5 credits.

Normalized the value to a value less than 1 and larger than 0 in the end for returning the similarity.

User: float similarity\_user\_p(user one, user two)

It’s based on how much same item and similar item between two users. Same item give credit 1, similar item give a partial credit which is a value less than one and larger than 0. It depends on the distance between these two items.

Same artist or genre:

Because artist and genre itself is just an id (string), both of them have no attributes, same id is 1, different id is 0.

**3. Implementation**

I use C++ STL to implement which I’m familiar with. STL container vector is used a lot to maintain the user information, track information, album information, etc. I also used a lot of STL algorithms such as sort, binary search, union, intersection when find the association between uses, tracks, albums.

**4. Result**

The result can be found in the /project09/result/validation\_predicted\_results.txt.