**User model**

There are four part in user model, 1) “liked songs” storing all the song’s ID that a specific user have given “like” reaction; 2) “hated song” storing all the songs” ID that a specific have given the “hate” feedback; 3) “mood” representing the user’s current mood in form of vector which ,in our case, has 6 dimension that are “happy”, “sad”, “angry”, loving”, “”,””, respectively. Each domain of element in this vector is [0, 1] represent the user’s preference on the mood of the songs from not interest to high interest; 4) “song category preference” representing, in form of vector, the user’s preference on the category of songs, specifically, the percentage of each song category to listen in an exact moment. The sum of the vector should be 1. In our application, there are 3 dimensions in this vector representing preference on “pop”, “jazz”, “rock”, respectively.

There are two important assumption to highlight here. The reason, instead of combining the “song category preference” and “mood” as one vector, we separate the “mood” and “song category preference” is because we assume that the preference of song category is stable, but the user’s mood is unstable. As the consequence, in the user model updating phase, there should be different strategy to update the user model, the “mood” to update flexibly and the “song category preference” to change slowly.

**Songs tags**

Each song has two types of tag, 1) tag of category, in specific, “pop”, “jazz”, “rock”; 2) tags of mood, in specific, are “happy”, “sad”, “angry”, loving”, “”,””. This type of tag could be multi-tags, though for the consideration of simplification of implementation, we tagged each song’s mood only once.

**Song matching**

Given the user’s model and a database storing all the manually-tagged songs, the songs match model recommends a song to the user following step listed below.

1. Weighted-randomly pick the song category to recommend. For example, the “song category preference” in user model, say, is [0.5, 0.2, 0.3], which means the users like pop most, followed by rock then jazz, then there are 50% probability for the pop to be chosen, as well as 20% for jazz and 30% for rock.
2. Score every song belonging to the specific category picked in last step. The strategy of scoring is based on the mood of user and song. The mood of user and song should be vectorized first. The song’s mood vector is a 6-dimension vector with each element set to 0 except the one relative to the mood tag set to 1. For example, a song tagged happy should be vectorized as [1, 0, 0, 0, 0, 0]. The score of each song is basically calculated by cosine of a user’s mood vector and song’s mood vector, which represent the similarity between 2 vectors (Steinbach, M., Karypis, G. and Kumar, V., 2000, August. A comparison of document clustering techniques. In KDD workshop on text mining (Vol. 400, No. 1, pp. 525-526).
3. Weight-randomly pick the songs scored. The higher the score is, the higher the probability to be chosen.
4. If the song picked is inside the user’s “hated songs”, go back the step 3.

**User model updating**

In our application there are 4 reactions: “no action”, “like”, “skip”, “hate”.

When the user gives “no action” which means that the user listens the song from the begin to the end. We assume for this user the song sounds good. When the user gives “like” reaction, we assume that this user must have some strong feeling on this song from which the information about the taste and preference of this user could be mined. These 2 reactions are considered as positive reaction.

For the “skip” reaction or “hate” reaction, we consider it as a negative reaction. We assume that the user would not like to be recommended the song “hated” before, and sometimes the category and mood are both fine, but users somehow hate this song.

Based on the reaction of a user after listening a song every time, we know a bit more about this user so that to recommend more precisely in the song matching phase. There are 4 parts of user model to introduce one by one.

“liked songs”:

The song’s ID will be stored in “liked songs” in user model if user give the “like” reaction.

“hated songs”:

The song’s ID will be stored in “hated songs” in user model if user give the “hate” reaction.

“song category preference”:

The song category preference is initialized when signing up, user order their preference on different song categories which give use a prior probability of recommendation. There should be a warm-up phase before phase before the number of songs liked meet the threshold we set. After that, the “song category preference” is ready to be updated. Each time the user gives a “like” reaction, the “song category preference” are recalculated. Each element in the “song category preference” vector assign with each liked song in relative category divided by the number of liked songs. Only if the warm-up phase is long enough, the cardinal number of liked song is big enough which make the “song category preference” vector stable.

“mood”:

Since we assumed users’ mood are unstable which means their preference on songs with different moods are unstable. In recommendation system, there are a problem called “stability vs. plasticity problem” (Billsus, D. and Pazzani, M.: 2000. ‘User Modeling for Adaptive News Access’. User-Modeling and User-Adapted Interaction 10(2^3), 147^180). To solve this problem, A common approach is to import a discount rate to lower the influence of older information, as a trade-off, at the risk of losing information about the long-term interests.

The method we take is, basically, to let the history information about the user’s mood influence the changing of current mood by abstracting the stability of their preference on songs with different mood. For example, if the user is moody person, every time he/she see a tragedy movie, he/she would like to listen some sentimental music that usually he/she would not like to listen. The moods of the “liked songs” in this person’s user model will be highly fluctuated. So, by calculating the standard error of each element of mood vector in each song in “liked songs”, we measured the stability of this user’s mood, which is the updated rate of the user model.

So, every time we update the user’s “mood”, if the user gives positive reaction to a specific song, lower the difference between this song’s mood vector and user’s mood vector multiplied by updated rate in form of standard error. if the reaction is reaction negative, vice versa.