2016 IEEE 22nd International Conference on Parallel and Distributed Systems

Smart-DJ: Context-aware Personalization for Music Recommendation on Smartphones

Chengkun Jiang*, Yuan He*
*School of Software and TNLIST, Tsinghua University, China

***Abstract*—Providing personalized content on smartphones is significant in ensuring user experience and making mobile applications profitable. The existing approaches mostly ignore the rich personalized information from user interaction with smartphones. In this paper, we address the issue of recommending personalized music to smartphone users and propose Smart-DJ. Smart-DJ incorporates an evolutionary model called Incremental Regression Tree, which incrementally collects contextual data, music data and user feedback to characterize his/her personal taste of music. An efficient recommending algorithm is designed to make accurate recommendations within bounded latency. We implement Smart-DJ and evaluate its performance through analysis and real-world experiments. The results demonstrate that Smart-DJ outperforms the state-of-arts approaches in terms of recommendation accuracy and overhead.**

***Keywords*-Context-awareness; Personalization; Recommendation; Incremental Regression Tree; User Feedback;**

I. Introduction

With fast development of communication technologies and proliferation of mobile applications, smartphones are replacing desktop computers as the major source of clients to access content over the Internet. Convenient and ubiquitous network connections on smartphones enable users to enjoy a variety of content anytime and anywhere. The content of users' interest may be entertainment, leisure, sociality, business, learning, and so on. Among those applications providing such content, music listening is a typical example that attracts countless users.

Listening to music in mobile contexts, however, introduces challenges to both smartphone users and application designers. Selecting a song to play usually requires the user's attention and several operations on the screen. Doing this in a stationary state is easy, but becomes fairly inconvenient when the user is in a mobile state, e.g. walking, exercising, and driving. As users get tired of repeating those preset playlists, enjoying recommended music from the Internet is undoubtedly an attractive experience. Nevertheless, a user under different contexts is likely to have different preferences of music. One may want some relaxing and soft music when he/she is resting, while may prefer energetic songs when he/she is having sports. More importantly, different users tend to have different taste, leading to different preferences while listening to music, even they are under the same context. Can a music recommender be smart enough to understand a user's personal needs anytime and anywhere? That becomes a crucial and challenging issue.

Various approaches have been proposed to tackle the issue of music recommendation, including early works proposed for desktop applications. A category of existing approaches called Collaborative Filtering (CF) [1] assume similar people share similar interest and recommend widely welcomed music, failing to satisfy the personal taste of users. Another category called Content-Based approaches (CB) [2, 3] explore the similarity among music and make recommendations according to a user's listening history, neglecting the changes of listening preferences under different contexts. Recent studies show that the rich sensing capability may produce more indication to a user's real-time listening preference [4–9]. Those approaches exploit a user's contextual information to describe user's real-time state. According to such information, they recommend music that other users in similar contexts listened to, which follows the idea of CF. So even with more contextual information, those approaches still face the difficulties in meeting a user's personal needs.

According to the above facts, we find that personalizing recommended music with context awareness is a promising solution, which in turn means critical challenges in three folds: First, a personalized music recommender must be built on large amount of information from context sensing and music listening history. Processing and managing those information is clearly a non-trivial task for resource-constrained smartphones. Second, decision making in the recommender must be sufficiently fast, not hurting the listening experience in terms of waiting time. Third, like many smartphone applications, music listening is an interactive process. It is significant for a music recommender to consider users' feedbacks. Both the explicit feedback (e.g., user's rating on music) and the implicit feedback (e.g., user's listening behavior during listening) offer useful information. How to utilize them to further improve the recommendation accuracy remains an open problem.

In order to address the above challenges, we propose Smart-DJ, context-aware personalization for music recommendation on smartphones. Based on the rich sensing capability on smartphones, Smart-DJ builds a personalized light-weight model called Incremental Regression Tree to map heterogeneous user contexts to music features. The model

1521-9097/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPADS.2016.25

is able to evolve with the listening history, so as to provide better and better characterization of personal music taste. The recommending algorithm based on the model is highly accurate and efficient, preserving user experience during listening.

Our contributions can be summarized as follows:

- We propose the model of Incremental Regression Tree to capture user's music preferences in different contexts and incorporate user's both explicit and implicit feedback into the model. The model can evolve to satisfy user's personal music requirements. Moreover, it has fixed maximum height and low cost of maintenance.
- We devise an efficient recommending algorithm that utilizes contextual information to reflect a user's real-time state and needs. The algorithm provides in-time music recommendation with personalized tastes.
- We implement Smart-DJ and evaluate its performance through analysis and real-world experiments. The results demonstrate that Smart-DJ outperforms the state-of-arts approaches in terms of recommendation accuracy and overhead.

The rest of the paper is organized as follows. Section II discusses the related works. In Section III, we elaborate on the model of Incremental Regression Tree. Section IV introduces the recommending algorithm. In Section V, we describe in detail the implementation of Smart-DJ. We theoretically analyze the complexity in terms of time and present the experiments and the evaluation results in Section VI. Section VII concludes the paper and discusses the future work.

II. Related works

In this section, we survey existing research on music recommendation.

Traditional music recommendation system can be classified into three categories:*collaborative filtering* (CF), *content-based* (CB) techniques, and hybrid methods [10]. CF uses information of similar users to predict the target user's interest. The target user is recommended with the songs other users that have the similar listening history or music searching history like [1]. However, music preference is subjective. So the assumption behind the CF method that users with similar listening behaviour have similar taste on music is vulnerable. CF also suffers that it can hardly recommend a new song that no user ever listened to. Different from CF, CB methods try to discover music similarity based on their audio or signal information and recommend users the songs similar to their previously-listening ones [2, 3]. To some extent, CB solves problems in CF, but it is still an active area that how to measure music content and the recommendation solely based on content similarity ignores the dynamics of listening states. Hybrid methods aims to combine different models to increase the overall recommendation performance by weighting, cascade, mixing [11].

With the development of smart devices and the increasing availability of rich sensors on those devices, context-aware music recommender has attract more and more attention recently and provides a novel way to accurately customize personalized music recommender system [12, 13]. There are studies that try to formally define what is mobile context [14, 15]. In a word, context is everything about you or the environment where you are. So involving context into the recommender will improve the accuracy of inferring user's preference.

Most existing context-aware music recommenders combine the context with the CF [6–8, 16] or CB [4, 5, 9] methods to recommend music. Lee $et\ al.$ propose a context-aware recommender by case-based reasoning [6]. It collects user contexts such as time, weather, place and temperature, then recommends the user with the music that other users with similar contexts listened to. SuperMusic [16] recommends music to the user based on other users' listening history in same location. Rho $et\ al.$ similarly adopt the CF methods considering user's mood [7]. Su $et\ al.$ consider contexts such as location, motion, calendar, environmental conditions and health conditions [8]. It combines the context similarity with the listening content similarity as to improve the CF method. These systems take advantage of mobile contexts to describe user's state better than web, but still face the limitation to model the individual user's preference on different contexts.

Wang $et\ al.$ propose a probabilistic model to evaluate the possibility the song is suitable in the specific activity [9]. It depends on the autotagging and implicit user feedback to calculate the possibility that a song matches the context. Every time a new song listened or added, the whole playlist ranking needs to be recalculated. Cai $et\ al.$ extract music textual meta-data with emotional information and form user's context based on the emotional word terms, so the recommendation is based on the music that other users like in the similar emotional contexts [5]. A single label such as activity or searching keywords of user's current state may not accurately reflect user's actual requirements. The description of music with meta-data can be different among different users. They still suffer the problem of personalization.

To associate songs with context, most context-aware recommenders use manually supplied metadata, labels and ratings [17–20], which is dependent on common description of different users. We use the audio features that independent of other users' opinions to better represent a song. To the best of our knowledge, no existing context-aware music recommenders try to model user's music preferences on the objective music audio features in different contexts.

III. Incremental Regression Tree

As mentioned before, music recommendation in mobile environment puts extra limitations on resources and user

Table I
COLLECTED CONTEXT

Category	Context Type	Comments	Range of discretized values
Activity Level	Acceleration	Acceleration in three directions	3
Noise	Microphone	Environment noise level	5
Time	Time of the Day	Time you listen to the music	6
Social Contact	SMS Frequency	The frequency of SMS usage	3
	Call Frequency	The frequency of making a call or receive a call	3

Table II
EXTRACTED AUDIO FEATURES

Category	Comments
Tempo	Reflect the rhythm of the song
Pitch	Reflect the melody of the song
MFCC	Reflect the frequency distribution on Mel scale and we select the first four coefficents

experience. Complex algorithms such as SVM, deep learning or heavy user involvement for model training are not suitable. So we propose a music recommender in smartphones, which incorporates the light-weight *Increment Regression Tree* (IRT) [21] that incrementally adjusts the model to reflect individual user's diverse tastes in different situations. The way of increment guarantees the performance with smaller training samples, because the accuracy of recommendations will be improved with the recommender evolving. IRT matches user contexts automatically with music features and contexts are organized hierarchically so that each combination may implicitly refer to a situation. We will present the IRT in detail below.

A. Overview of the Model

We try to directly map the collected contexts to specific music features through the IRT model. There are five contexts we consider that are mostly related to user's music preference as described in Table I and we discretize these contexts into 3-6 value levels respectively. As for music features, we extract the relatively stable audio features to represent music appropriately [22]. The audio features we select are shown in Table II and the features of each song are pre-processed to store in the server. The details of these information process are presented later in Section V. It is true that users music predilection can change over time, so only considering audio features can hardly respond to the change. To make up for the flaw, we take user feedbacks into consideration, which can be a direct indication of music preference under certain state.

When a user is listening to a song, we will collect three types of data: the listening contexts C; the audio features of current song F and the user feedback on that song $Rate$. We denote these as a piece of record $R(C, F, Rate)$ and the IRT is constructed incrementally with these records one by one. It is noticeable that sometimes it is a certain specific feature that affects user's taste. So we build different IRTs for different audio features to ensure the dominant feature is correctly identified. A single IRT reflects user taste on a single audio feature f in F. An example of IRT on a single feature is shown in Fig. 1. Table III provides the symbol reference.

Table III
SYMBOLS

Symbol	Meaning
C	All the contexts we collected
F	All the audio features
$Rate$	The final rating for a song
$R(C, F, Rate)$	A record of a song
c	One context that belongs to C
f	One feature that belongs to F
$m_i(f)$	The value of feature f in ith record
$E_n(f)$	Entropy over n songs about the feature f
$E_{parent}(f)$	Entropy of the parent cluster about the feature f
$E^c_{v_i}(f)$	Entropy of the records that the value of c context is v_i
$mean_n(f)$	The mean value of feature f over n songs
$I_c(f)$	Information gain with context c
n_c	No. of values on context c
N^c_{vi}	No. of records that the value of c context is v_i
N	No. of records in the cluster
ϵ	The threshold for the entropy

B. Tree Components

As shown in Fig. 1, there are two parts in the IRT. One is the circle node called the splitting node and each node splits the records in terms of the context value. Another part is the rectangle node called the leaf node that contains the records with similar audio feature. The structure of the IRT can evolve with the number of listening records. The evolution of structure proceeds in a light way with small overhead of computation, which we will mention in next section.

Splitting Node: We can see in Fig. 1, every splitting node has an associated context in its circle. The associated context is selected based on the music features. For example, if the user listens to fast rhythm music in the afternoon and slow rhythm music at night, the time context will be selected as the associated context to form a splitting node. Then two kinds of music will be classified into two branches of the node corresponding to the value of the time context. If the features in one branch still varies much, then the IRT will select other informative context to form a splitting node recursively. For example, in the Fig. 1, time context is first selected to split the music records and then acceleration is recursively chosen to be a sub-level context in the top branch. The algorithm of how to choose the context and update the structure are presented in Section IV.

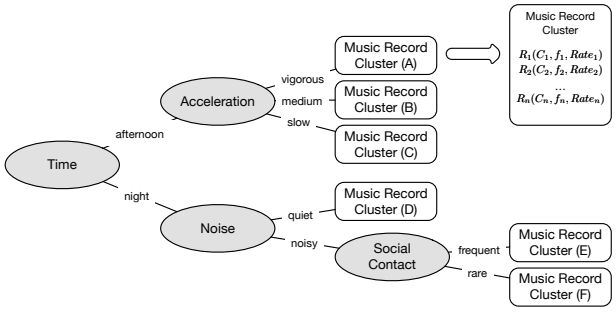


Figure 1. An example of IRT for music recommendation.

Leaf Node: The leaf node each contains a set of music records $R(C, f, Rate)$ and f is one feature of audio features F. In each record set, all the values of feature f gather together in a range that is controlled by a threshold. The threshold is defined in terms of value variation around their average value. The higher the threshold is, the broader the feature values in one cluster will distribute and the more flexible music taste prediction will be. On the contrary, if we set a small threshold, the values in one cluster will converge tightly to make the prediction accurate. We can find there exists a tradeoff between flexibility and accuracy in threshold selection. In each cluster, we can also find that all the music records share some same context value. Take the cluster in the bottom branch of "social contact" for example. All the music records have same context values night, noisy, rare. However, other contexts such as acceleration may have different values.

IV. Model Training and Recommendation

As we mentioned before, the IRT model is trained in an incremental way to capture a user's music preferences. It automatically organizes the hierarchy structure of contexts so that each record cluster may refer to user's certain music predilection. The mobile environment further limits the complexity of computation and storage, so the incremental update of IRT should be efficient and simple. We will first present how the IRT incrementally is trained to capture user music predilection and then how we exploit the IRT to make music recommendation with consideration of user feedbacks.

A. Incremental Training

Every time a new record is observed, it will be put in the corresponding record cluster based on its context values. If the feature difference exceeds the threshold, the algorithm should find an appropriate context to form a splitting node to split the records. This process is then recursively done in the split record clusters until the differences in every cluster are below the threshold.

1) Feature Difference: To determine when to split the records, we use the *Entropy* to indicate the value difference. When the entropy exceeds a defined threshold ϵ for an audio feature, the IRT needs to form a splitting node. In Section VI-B, we set appropriate defined thresholds for different features. We use variation to represent entropy. The smaller the entropy of a record set is, the more similar the audio feature values of the record set are. If the entropy is large, it is likely to form several value clusters of the audio feature in the record set. So we check the context values to find which context affects the difference. The formula to calculate the entropy is below:

$$E_n(f) = \sqrt{\sum_i^n (m_i^2(f) - mean_n(f))/n}$$

$$mean(f) = \sum_i^n m_i(f)/n$$

If we now have the entropy over $n-1$ songs $E_{n-1}(f)$ and the mean value of f is $mean_{n-1}(f)$. When a new record comes, we can obtain $E_n(f)$ and $mean_n(f)$ based on the formula below:

$$\begin{aligned} E_n(f) =& \frac{n-1}{n}(E_{n-1}(f) - mean_{n-1}(f)) \\ &+ \frac{m_n^2(f) - m_n(f)}{n} \end{aligned} \tag{1}$$

Then we update the mean value as:

$$mean_n(f) = \frac{mean_{n-1}(f) * (n-1) + m_n(f)}{n} \tag{2}$$

It can be observed that the entropy update can be computed in an incremental way within the constant time.

2) Context Selection: After determining when to split records, the IRT needs to find a most appropriate context to split the record set. We use the *Information Gain* to select the context. It is the reduction of entropy when select one context to split the records. The higher it is, the better the split is. We can calculate the information gain on the specific context c considering feature f as:

$$I_c(f) = E_{parent}(f) - \sum_i^{n_c} \frac{N_{v_i}^c}{N} E_{v_i}^c(f) \tag{3}$$

The first part in the equation represents the entropy of cluster before split and the second part is the weighted entropy of all the clusters split based on the context c. So the information gain measures the reduction of the entropy. We can find all the computation can be finished in constant time, which is of great importance in mobile environment. The algorithm to select the context is presented in Algorithm 1. The arrays E_{ij},num_{ij} and $mean_{ij}$ in the algorithm is the entropy, the number and the mean value of the music with the context i to be the value j.

Algorithm 1 Context Selection

Input: Context vector C; Feature f; Node entropy E_{node}; Maintained array E_{ij}, num_{ij} and $mean_{ij}$; Defined threshold ϵ;
Output: The decisive context c

```
E_node = updateEntropy(E_node, f)
for all c_i in C do
    E_{ic_i} = updateEntropy(E_{ic_i}, f)
    mean_{ic_i} = updateMean(mean_{ic_i}, f)
end for
if E_node < ε then
    c = nil
else
    maxI = 0
    c = nil
    for all Context Type k do
        I_k = calcuInfoGain(E_node, E_kj, num_kj)
        if I_k > maxI then
            maxI = I_k
            c = k
        end if
    end for
end if
return c
```

$updateEntropy$ and $updateMean$ correspond to the formula (1) and (2), $calcuInfoGain$ corresponds to formula (3).

3) Tree Update: In the training process, every time a new record comes, the $ContextSelection$ is called to find the context to split current music records. If the return is nil, we just add the record. Otherwise the node is a splitting node. If the returned context is same as the context of the splitting node, we pass the record to the branch based on its value of this context. Then in this branch we do the process recursively based on the music records belonging to this branch. If the returned context is different, which means the previous context is not the most informative one, we update the tree structure to associate the new context to the node. Then we generate branches to split the music records and do the above process recursively in each branch.

B. Music Recommendation

The music recommendation is based on the audio features inferred through the IRT when the current contexts are available. We will show how to make the inference with involvement of user feedback.

1) User Feedback: The system will collect the user feedbacks when user is listening to the recommended songs and there are two kinds of feedback can be used when each song is played.

Explicit feedback: We set a user rate bar of typical five-level in our Smart-DJ. Users can give scores from 1 to 5 representing from strongly disagree to strongly agree when a song is listened to. It needs to be considered that users tend not to give a low rate unless they really dislike the song, so we set 3 points as the default rate.

Implicit feedback: Sometimes users may not trouble themselves to rate the song, but still we can infer the users' preference on the song using some implicit feedbacks. We find users tend to change the song quickly when they don't like the song. Thus we calculate an implicit rate on the listening time: $Rate_{implicit} = maxRate\frac{t_{listening}}{T_{song}}$, where $maxRate$ is the highest rate of explicit rating, $t_{listening}$ captures the user's listening time and T_{song} defines the duration of the music that can be decoded when the music is played.

The final rate in the record is the combination of the two kinds of feedback, which is $Rate = \alpha Rate_{explicit} + (1 - \alpha)Rate_{implicit}$. α controls the ratio of each feedback.

2) Feature Inference: When we observed the current contexts of a user, we want to make the inference on the audio features. Three situations will be meet in the inference.

Initial Step: No records have been observed to form an IRT. Due to the intrinsic evolution of the IRT, we randomly select typical songs of different music genres and keep the songs with high user rate to build the IRT. We also support users to manually select songs when no recommendations are suitable.

Normal Process: When the current contexts are available, the algorithm searches the IRT for the corresponding record set that reflects user's preference. Then the weighted mean method is exploited to consider the user feedbacks, The final inferred feature is the weighted mean of the features in the cluster:$f = \sum_i^n rate_i f_i / \sum_i^n rate_i$.

Exception: There are situations that the specific new context value can't match any branch of the current splitting node. Based on the assumption that a user shares similar music preference when most contexts are the same and the fact that the context at higher level in the tree is more informative, we use the record clusters that have some same contexts to infer the audio feature. For example, the current contexts are night(time), medium(noise), frequent(social contact), then we can't find any corresponding record cluster in Fig. 1. We use all clusters in the left subtree to get three features by normal process. Then we weighted the features based on the number of same contexts to get the predicted feature: the left bottom cluster has 2 same contexts and others both have 1. $f = \sum_i^n num_i f_i / \sum_i^n num_i$

After getting the features from different IRTs, we upload the features to the cloud to fetch the suitable music from the server music database.

3) Cloud Music Match: The purpose of the music match module is to accurately and quickly find pieces of music that have the similar features to the required features. In our system, we adopt the data structure *Vantage-point Tree* (VP Tree) [23] to cluster the music based on the cosine distance

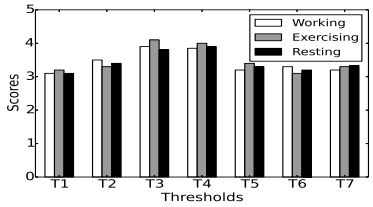


Figure 2. Performances for Different Thresholds

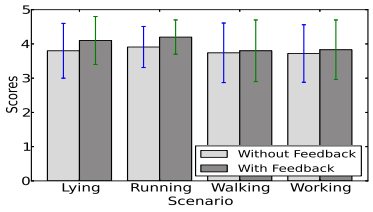


Figure 3. Performance in DifferentScenarios

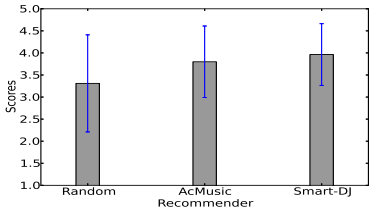


Figure 4. Overall Rates

of their audio features. It will take $O(logN)$ time to fetch the similar songs given the received features.

V. System Implementation

Our system contains two parts: the recommender in the client and the music match server. The recommender collects users' contexts to predict music features with the IRT. Then the features will be uploaded to the server to match the songs. Finally, the recommender receives the songs for the user and obtains the user rates when the songs are played to form records for IRT updating.

The contexts we collected are presented in Table I. We use the linear magnitude of acceleration to represent user's activity level. Since the acceleration data is sensitive, we collect 20 seconds data processed by a low pass filter and calculate the average value for each 4 seconds to get 5 discretized values. We then select the value with majority voting. We collect the data for the next song when the previous one is played and for the first song, we collect 2 seconds data. We classify the noise into 5 levels based on the decibel of noise that can be detected with the microphone equipped on the smartphone. To reduce the impact of noise, we collect 10 seconds of noise amplitude and average them to ensure an accurate noise level. The time of a day is divided as early morning, morning, noon, afternoon, evening, late night. To collect user's phone usage, we set a time trigger to collect every half-hour user's message number and call number to calculate the frequency.

In our system, we use acoustic features that is independent of the human work. The features we select are tempo, pitch and the first four Mel-frequency cepstral coefficients (MFCCs). To extract these features from the audio file, we first divide the whole audio into fixed-time segments. Half time of the segment overlaps with the one before it. Then we run different feature extraction algorithms on each segments. Because the start and the end part of the audio contain little information about the music, we ignore several segments at the beginning and the end of the audio and extract the tempo, pitch, MFCCs for each segment. In order to form the final features vector, we calculate the means of the tempo, pitch and first four coefficients of MFCCs.

We construct a dataset of 876 songs crawled from different music class in music websites. We crawled them and run our feature extraction program to process them. We find 16 volunteers that are Android smartphone users for our evaluation and they are graduate or undergraduate students. The number of males and females is equal and their ages are between 21 and 28 years old. All of them listen to music in different situations during the day.

VI. Evaluation

A. Energy Evaluation

We measure the recommendation latency and the total power consumption to run the application on a HTC M8 android smartphone. For the recommendation latency, we focus on the latency between user clicking listening button and the recommendation generated without the network latency. We calculated the three latency: the network latency, the cloud responding latency and the total latency. The network latency includes feature sending and audio file fetching. The responding latency refers the time the cloud uses to find proper songs. We computed an average value of them with 20 songs listened to. The average total latency we obtained is 623 ms with 234 ms of the network latency and 11 ms of server responding latency. So the prediction latency is 623-234-11=378 ms, which has almost ignorable influence on the user experience.

To estimate the power consumption of our system, we collect the system power consumption and the phone total power consumption once an hour. The average power consumption is 113 mW and we noticed that peak power consumption can reach 600 mW when the sensors start working. However, the sensors working time only occupies a small portion of the listening time.

B. Parameter Tuning For IRT

We present the max value gaps of different feature values in Table IV. It is hard to find the optimal combination of different thresholds, since there are countless combinations of different thresholds for six features. We try to find a proper combination of thresholds for six features. 7 potential thresholds are selected for each feature with equal interval from 0 to half the max value gap. We form 7 combinations of thresholds for experiment and the thresholds of the six features have the same sequence in their own 7 values for

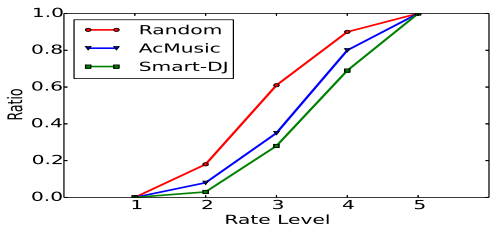


Figure 5. CDFs of Rates

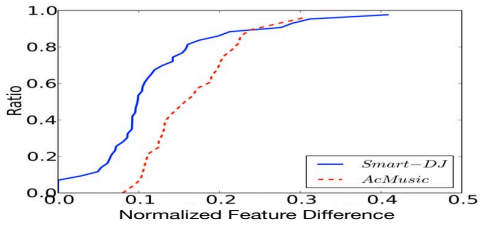
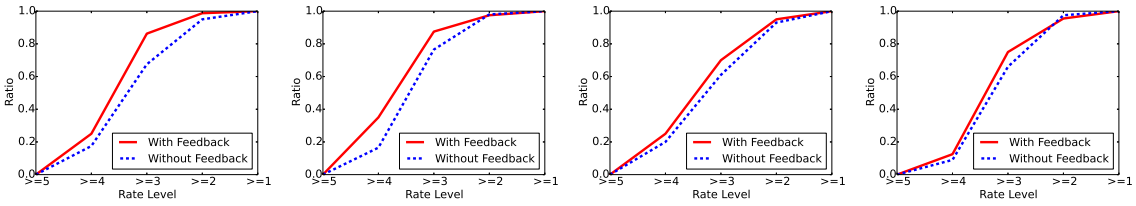


Figure 6. Normalized Feature Difference



(a) Lying (b) Running (c) Walking (d) Working

Figure 7. CDFs of user rates for Smart-DJ with and without feedback in different scenarios.

Table IV
MAX VALUE GAP FOR DIFFERENT FEATURES

Feature	Tempo	Pitch	MFCC			
Variation	94	573	10.6	4.8	1.6	1.5

every combination. We just need to experiment on the 7 combinations to determine a proper one.

We select three scenarios in which 12 volunteers have different music preferences and ask each volunteer to listen to 15 songs in each scenario. We record the total 45 songs for each volunteer to train the system 7 times with 7 different thresholds, then test the system performance with 20 songs recommended by the trained system in each scenario. We compute the average scores for each set of thresholds in each scenario with 5-points Likert scale from "strongly disagree"(1) to "strongly agree"(5). The 7 experiments are represented from T1 to T7 based in the ascending order. We demonstrate the results in Fig. 2 (d). Considering the overall ratings, we find the T3 and T4 have better performance than others. Since we hope the system to be more flexible, we use T3 to be the set of thresholds for the experiments next. When the thresholds are low, some unnecessary splitting nodes need to be formed and become the noise to affect the inference. When the thresholds are high, it will tolerate new type of feature values that should be detected as new branches.

C. Comparison

We compare the performance among three recommenders:1) random recommender that randomly selects songs to recommend, which provides a baseline; 2) The auto mode for the recommender presented in [9] that we call it AcMusic; 3) Smart-DJ. We select a random subset of 600 songs from the dataset and initialize it for AcMusic. All the participants are asked to listen to music in different scenarios including excising, working, resting, walking etc. It is required that in every scenario participants listen to 10 songs for each recommender and rate them using 5-points Likert scale. We collect the rating data of three recommenders in two days and demonstrate the average and standard deviation of the ratings in Fig. 4.

It is observed that AcMusic and Smart-DJ have better performance than random system with higher average rate and lower deviation. To further assess our system, we asked users to choose their preferred songs in each scenario and computed the normalized feature difference between the preferred songs and 10 songs recommended with two systems. We plot the CDF of the feature difference in Fig. 6. Besides, we collected user rates to draw the rate distribution in Fig. 5. Nearly 70% of the songs have feature difference below 0.1 through Smart-DJ and the ratio of user rates above 4 points in Smart-DJ is higher than others, which both consolidate the better performance of the Smart-DJ.

D. Multi-Scenario Performance Analysis

We can use the performances in different scenarios to assess the system. We select four scenarios that all participants will listen to music in: lying in bed before sleeping, running for exercise, walking in the street, working in the office in the daytime.

We experiment on Smart-DJ with and without feedbacks. We demonstrate the results of the rating distributions in different scenarios in Fig. 7 and the overall average ratings and deviations are presented in Fig. 3.

In Fig. 7, we can find that Smart-DJ with feedback has more ratings above 3 points and less below 3 points. From the Fig. 3, we find that the average rating points with the feedback in five scenarios are all around 4 with the standard deviation less than 1. Although the no-feedback version has average ratings around 3.8, the user feedbacks can provide significant information on user music preferences. The high average points and low deviation indicate

VII. Conclusions

In this paper, we present a novel personalized context-aware music recommender, Smart-DJ, which effectively utilizes various contextual information that can be collected with off-the-shelf smartphones. It builds the model mapping from user contexts to music audio features for personalized recommendation. It also takes into account users' explicit and implicit feedback to adjust the model for recommendation accuracy. In order to better meet users' experience, we make recommendation according to the most possible music features in the current contexts rather than the ranking of every single possible music. Smart-DJ is an accurate, efficient personalized recommender with low overhead that is suitable for smartphones.

VIII. ACKNOWLEDGMENT

This work is supported in part by National Natural Science Fund of China for Excellent Young Scientist under grant No.61422207 and the research fund of Tsinghua - Tencent Joint Laboratory for Internet Innovation Technology.

References

[1] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*, pp. 291–324, Springer, 2007.

[2] Q. Li, B. M. Kim, D. H. Guan, *et al.*, "A music recommender based on audio features," in *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 532–533, ACM, 2004.

[3] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*, pp. 325–341, Springer, 2007.

[4] M. Braunhofer, M. Kaminskas, and F. Ricci, "Location-aware music recommendation," *International Journal of Multimedia Information Retrieval*, vol. 2, no. 1, pp. 31–44, 2013.

[5] R. Cai, C. Zhang, C. Wang, L. Zhang, and W.-Y. Ma, "Musicsense: contextual music recommendation using emotional allocation modeling," in *Proceedings of the 15th international conference on Multimedia*, pp. 553–556, ACM, 2007.

[6] J. S. Lee and J. C. Lee, "Context awareness by case-based reasoning in a music recommendation system," in *Ubiquitous Computing Systems*, pp. 45–58, Springer, 2007.

[7] S. Rho, B.-j. Han, and E. Hwang, "Svr-based music mood classification and context-based music recommendation," in *Proceedings of the 17th ACM international conference on Multimedia*, pp. 713–716, ACM, 2009.

[8] J.-H. Su, H.-H. Yeh, P. S. Yu, and V. S. Tseng, "Music recommendation using content and context information mining," *Intelligent Systems, IEEE*, vol. 25, no. 1, pp. 16–26, 2010.

[9] X. Wang, D. Rosenblum, and Y. Wang, "Context-aware mobile music recommendation for daily activities," in *Proceedings of the 20th ACM international conference on Multimedia*, pp. 99–108, ACM, 2012.

[10] Y. Song, S. Dixon, and M. Pearce, "A survey of music recommendation systems and future perspectives," in *9th International Symposium on Computer Music Modeling and Retrieval*, 2012.

[11] R. Burke, "Hybrid recommender systems: Survey and experiments," *User modeling and user-adapted interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[12] M. Kaminskas and F. Ricci, "Contextual music information retrieval and recommendation: State of the art and challenges," *Computer Science Review*, vol. 6, no. 2, pp. 89–119, 2012.

[13] F. Ricci, "Context-aware music recommender systems: workshop keynote abstract," in *Proceedings of the 21st international conference companion on World Wide Web*, pp. 865–866, ACM, 2012.

[14] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*, pp. 304–307, Springer, 1999.

[15] G. Chen, D. Kotz, *et al.*, "A survey of context-aware mobile computing research," tech. rep., Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[16] A. Lehtiniemi, "Evaluating supermusic: streaming context-aware mobile music service," in *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pp. 314–321, ACM, 2008.

[17] H.-S. Park, J.-O. Yoo, and S.-B. Cho, "A context-aware music recommendation system using fuzzy bayesian networks with utility theory," in *Fuzzy systems and knowledge discovery*, pp. 970–979, Springer, 2006.

[18] S. Dornbush, A. Joshi, Z. Segall, and T. Oates, "A human activity aware learning mobile music player," in *Proceedings of the 2007 conference on Advances in Ambient Intelligence*, pp. 107–122, IOS Press, 2007.

[19] S. Cunningham, S. Caulder, and V. Grout, "Saturday night or fever? context-aware music playlists," *Proc. Audio Mostly*, 2008.

[20] M. Kaminskas and F. Ricci, "Location-adapted music recommendation using tags," in *User Modeling, Adaption and Personalization*, pp. 183–194, Springer, 2011.

[21] P. E. Utgoff, "Incremental induction of decision trees," *Machine learning*, vol. 4, no. 2, pp. 161–186, 1989.

[22] M. F. McKinney and J. Breebaart, "Features for audio and music classification.," in *ISMIR*, vol. 3, pp. 151–158, 2003.

[23] A. W.-c. Fu, P. M.-s. Chan, Y.-L. Cheung, and Y. S. Moon, "Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances," *The VLDB JournalThe International Journal on Very Large Data Bases*, vol. 9, no. 2, pp. 154–173, 2000.