Adaptive Jukebox: A Context-Sensitive Playlist Generator

Matthew Rizzo

Department of Artificial Intelligence, University of Malta

Abstract. Nowadays, a lot of users own large collections of music MP3 files. Manually organising such collections into playlists is a tedious task. On the other hand random playlist generation may not always provide the user with an enjoyable experience. Automatic playlist generation is a relatively new field in computer science that address this issue, developing algorithms that can automatically create playlists to suit the user's preferences. This paper presents our work in this field, where we suggest that playlist generators should be more context-sensitive. We also present Adaptive Jukebox, a context-sensitive, zero-input playlist generator that recommends and plays songs from the user's personal MP3 collection. Initial experiments suggest that our system is more accurate than both a random generator and a system that does not take context into account.

1 Introduction

The increase in popularity of music downloads, coupled with the increasing storage capacity of personal computers has given rise to users having large music collections on their computers. Manually organising such collections into playlists is a tedious and time-consuming task. On the other hand, playlists generated using random shuffle fail to satisfy the criteria of a good playlist. A good playlist is one that can adapt music to suit the user's current mood and the activity he is involved in [OK06], and caters for the user's desire for repetition and surprise [PRC00].

The solution to this problem lies in automatic playlist generation, a relatively new field in computer science concerned with the development of algorithms that can automatically create playlists to suit the user's preferences. This paper presents our research [Riz07] in the field of playlist generation. We suggest that playlist generation needs to make a greater use of context and user modelling and we propose the Adaptive Jukebox — a zero-input, context-sensitive adaptive playlist generator that recommends and plays music while the user is using his computer. We are still in the process of evaluating our system, but initial results are encouraging and suggest that for two out of three participants in a user experiment, our system not only performs better than a random generator but also performs better than a system that does not take context into consideration.

In the following section we will go over literature in the area of playlist generation as well as the context and user modelling fields. In section 3 we describe the

design of our system and in section 4 we discuss some implementation issues. In section 5 we describe our evaluation approach and present our initial results. Finally, section 6 contains some concluding remarks.

2 Literature Review

2.1 Playlist Generation

Early research in the field of playlist generation mainly focused on the notion of music similarity and on generating a sequence of songs (playlists) that satisfy some user-defined criteria [PRC00] [AT00]. Researchers then started experimenting with signal processing in order to define song similarity which allowed them to generate sequences of songs that sound the same [Log02] [AP02]. Logan [Log02] however suggested that there was more to playlist generation and stated that an ideal system would incorporate context, user modelling and user feedback.

Context-sensitive playlist generators include Affective DJ [HPD98] which recommends songs based on skin conductance, a feature that correlates well with emotion, and PersonalSoundtrack [ET06] which adapts the music to the user's pace while the user is walking. Other systems such as PATS [PE02] and Recommend 'n Play [AH06] use pre-defined contexts of use or allow the user to manually state what context he is in. The most interesting work in this area is that by Oliver and Kreger-Stickles [OK06] who developed PAPA, a framework that caters for a broader definition of context. However, the only system built on it, MPTrain, only takes into account the user's heartbeat and current pace.

User modelling in playlist generation was first introduced by Field et al. [FHM01] who suggested storing the user preferences in a long-term and a short-term user model. Pampalk [PPW05] uses a short-term user model that stores the songs that the user accepts or rejects during the current session, eventually using it to recommend songs similar to the accepted ones and different to the rejected ones. In Recommend 'n Play [AH06], songs that are accepted together in a playlist are considered to be similar while rejected songs are assumed to be dissimilar. Thus the system builds a long-term model of what the user considers to be similar songs. PAPA [OK06] has the most sophisticated user model which stores demographic data, user history and mappings between context and user preferences.

Playlist generators that adapt their recommendations to suit users, take into account user feedback and use it to update their user model. Pampalk et al. [PPW05] and Elliott and Tomlinson [ET06] learn that songs skipped by the user should not be recommended in the future. Recommend 'n Play [AH06] lets the user inspect the playlist before it is played, and allows him to add or remove songs from the playlist, thus learning which songs are liked or disliked.

We believe that playlist generation can be improved by giving more importance to context because most existing systems only use one source of context (such as the heart beat or user pace) or a defined context of use. We also think that it is important to develop a sophisticated user model that can learn user preferences

in various contexts. Methods of collecting user feedback, on the other hand, have already been addressed successfully in literature and therefore our focus will be on context and user modelling which are introduced in the following sections.

2.2 Context

Context is any information about the circumstance, objects or conditions surrounding a user that is considered relevant to the interaction between the user and the computing environment [RC03]. Context includes:

- *Mood*: Research indicates that music and emotional state are strongly related [HA02] [LO03]. Emotion can be detected in various ways including facial expression recognition [PR00], audio-based emotion recognition [IC05], recognition through physiological inputs [PVH01] or through mouse and keyboard [ZGDG03].
- *Activities*:Musical preferences also depend on the activities that a user is involved in [OK06]. For a computer user, the activity is strongly related to the programs that are open and the windows that have focus.
- *Social Context*: The style of music may also depend on who the user is with [AH06].
- *Physical Context*: Physical context includes location and time both of which can have an effect on user preferences. In fact, Oliver and Kreger-Stickles [OK06] suggest that preferences might depend on whether the user is at work or in the car and Andric and Haus [AH06] state that preferences might depend on the time of day.
- *Environmental Context*: Environmental context such as weather as well as light and sound levels [RC03] can have various effects on musical preferences, and probably vary from one individual to another. Some people's moods might be effected by the weather, which in turn effect musical preferences. Light and sound levels might give other indications as to what the user is doing, what time of day it is and the general atmosphere that the user is in or is trying to create.

2.3 User Modelling

Jameson [Jam01] states that when designing a user model we need to understand its functions, the properties it will store, the inputs and the techniques used to construct it.
For playlist generators the function of the user model would be to learn user preferences to be able to recommend songs to the user. In literature, user properties modelled include beliefs, goals, knowledge, level of expertise and user interests. In our case we need to store the user's musical preferences, as in Music IR systems where user models store songs that the user considers to be good or bad [HMI03] and the definition of similarity [Rol01]. User model inputs can be self reports, response to test items and naturally occurring actions. The latter is the most common approach in playlist generation (e.g. user skipping a song). User

modelling techniques include stereotypes, logic-based approaches as well as machine learning and Bayesian methods. The latter is probably the most applicable to playlist generation.

3 Design

Adaptive Jukebox is our zero-input, context-sensitive playlist generator that plays music from the user's MP3 collection while the user is using his computer. It operates in two modes: indexing mode and training and playback mode. The indexing process loads songs from the song repository (e.g. the user's hard disk) and extracts features that are stored in a song database and later used for recommendation. During the training phase the system allows the user to select songs and build playlists manually and learns the user preferences while during playback the system plays songs to suit the user's current context.

Figure 1 shows the design of our system. Our main hypothesis is that through our approach we can play music that the user likes and that the system performs better than a random generator, or a generator that does not take context into account. We also aim to develop a user-friendly application that does not require any input (other than that during the training phase) and a system whose performance improves over time.

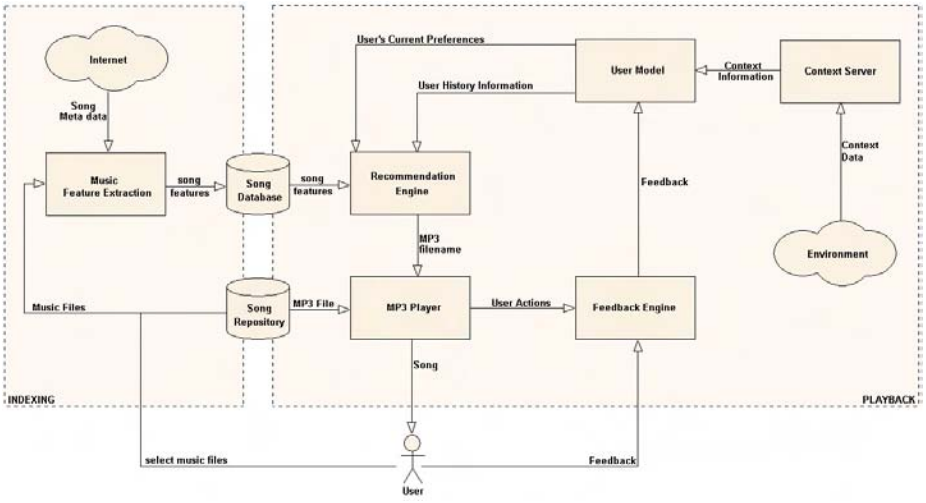

Fig. 1: Overview of Adaptive Jukebox

3.1 Music Feature Extraction

This module is responsible for indexing the songs in the user's MP3 collection and storing the data in the song database. The module first converts the MP3 file into a WAV file using JLayer[1]. The system then uses the libofa library to download the song title and artist's name from MusicDNS[2]. The WAV file and the song title and artist's name are then fed through a number of components that extract audio information and meta data respectively. Audio information includes loudness and fluctuation patterns (both extracted using CoMIRVA [Sch06]) and the beats-per-minute (which uses BeatRoot [Dix06] to detect the note onsets). Meta data includes the year the song was released (downloaded from www.musicbrainz.org), the lyrics, as well as genre and artist similarity. From the lyrics downloaded from www.lyricwiki.org, we first extract the song's language. We store lists of English, French, Spanish, German and Italian stopwords and count the number of occurrences of these words in the lyrics. The song's language is defined as the language with the highest count. For English songs, we check whether the song has explicit lyrics by passing the lyrics through a filter that detects bad words. Genre is detected by querying the Yahoo web service[3] for *artist + music + genre*. The summaries of the first ten documents are extracted. The number of occurrences of ten predefined popular genres is then calculated. The genre with the highest occurrence is taken to be the song's genre. We also use the Amazon web service[4] to detect the song's genre and also download artist similarity. Two artists are considered similar if their work appears in each other's recommended products on Amazon.
We therefore extract eight features (loudness, beat, fluctuation patterns, year, language, explicit lyrics, genre and artist similarity). The music feature extraction module then stores these features in the database. It also creates and stores a song similarity matrix that defines the similarity between all songs, along the eight features. So for every pair of songs, we store eight values that define the level of similarity between the songs.

3.2 Context Server

The context server is responsible for reading data from the environment and then detecting the user context. We read data from nine sources (applications that are open, mouse activity, keyboard activity, facial expressions, time, day of the week, brightness level, sky conditions and temperature).
For applications we record all windows that gain focus and during training we cluster this data using an agglomerative hierarchical clustering algorithm so that we can define the windows that make up the various activities that the user is involved in. As suggested by Oliver et al. [OSTS06], the similarity function used for clustering, is based on the times the windows were accessed and the titles

[1] www.javazoom.net
[2] www.musicip.com
[3] www.yahoo.com
[4] www.amazon.com

of the windows. During playback, the windows that have focus are compared to the application clusters in order to determine the current application context.
For mouse we record data such as acceleration, deceleration, click number, overshot number and overshot length [Mae05], average speed, efficiency and uniformity [Mae05], movements per minute, distance travelled per mouse movement and per minute and active time per minute. Data recorded for keyboard activity includes keystrokes per minute, average keystroke duration, and the number of different keys pressed. During training, both mouse and keyboard data are clustered using K-means into five categories, each of which defines a different pattern of mouse or keyboard activity. During playback, the system uses these clusters to determine the current mouse and keyboard activity.
Facial expressions are detected by first extracting 22 facial features from webcam snapshots using FSearch [CC06]. We then normalize the points read and calculate distances between points (such as between the eyebrow and the eye). During training we calculate the averages of these readings and then compare each reading to these averages to detect ten action units using rules that are similar to those used by Pantic and Rothkrantz [PR00]. For example, we detect that the distance between the left eyebrow and eye is greater than average and therefore signifies a particular action unit. During training we then cluster the action units into three categories using K-means. During playback the detected action units are compared to the clusters to determine the user's facial expressions.
The algorithms to detect the other types of context are simpler. The time and day of week are read from the system clock, the brightness level is the average RGB value of a pixel from a webcam snapshot and the sky conditions and temperature are downloaded from a web service.

3.3 User Model

Although we experimented with probabilistic user models, the best approach was to use context clustering. During training, the nine context values detected by the context server are clustered using a hierarchical clustering algorithm to determine the various scenarios the user is in. The songs played in each context cluster are then analysed and used to build the user model. We store two types of user properties for each context: the weights in the similarity function (that define how the similarity between two songs is computed given the similarity in terms of the eight features) and the list of songs that the user likes to listen to in that context. The weights in the similarity function are initially learnt through a genetic algorithm whose fitness functions outputs a high value if two songs played consecutively are considered similar by the function. The weights are then updated using an algorithm that is similar to the one proposed by Rolland [Rol01]. The preferred songs is a list of song-value pairs where the song at the top of the list (highest value) is the one that is played often in that context.
We also store the user history and a set of filters that determine whether songs by the same artist should be repeated in a session or whether the system should repeat songs. These two pieces of data are not dependent on context.

3.4 Recommendation Engine

Given the user preferences in the current context, the recommendation engine outputs the song to be played next. The system first creates an ordered list of suitable songs depending on the user history, the preferred songs in the current context and the songs that were accepted and rejected during the current session (Songs that are similar to accepted songs are given a higher position in the list than songs similar to rejected songs). The recommendation engine then outputs the first song on the list that does not violate one of the filters in the user model.

3.5 MP3 Player

The MP3 player plays the recommended song and reports any user actions to the feedback engine. The interface used to play songs is jlGui[5] because it is user-friendly and sophisticated.

3.6 Feedback Engine

When the user accepts or skips a song during playback, the feedback engine updates the user model. If a song is accepted, it is added to the list of preferred songs for the current context and the weights in the similarity function are updated to reflect that the current song and the last song played are similar. On the other hand if the song is rejected, the song's value in the preferred song's list is decreased and the weights in the similarity function updated so that they reflect the fact that the current song and the last song played are not similar.

4 Implementation

The system is implemented as a Java application so that it is highly portable. The only components that are dependent on the operating system or the hardware are the context server and the interface. The context server is divided into two components such that the component that reads data from the environment is not part of the Java application. It is in fact implemented as a .NET application and runs on Windows. The .NET application communicates with the main Java program by sending XML messages over TCP/IP. Having said that, the system can easily be ported to another operating system or hardware setting. For example, by developing an interface that plays music on a car MP3 player and by creating a system that detects a drivers's context, the system can be used in a car environment.

[5] www.javazoom.net

5 Evaluation and Results

We are currently evaluating the system on three participants who volunteered to use the system for a few weeks. The evaluated system is as described above, although we do not take into consideration facial expressions and brightness due to the unavailability of webcams. During the first weeks the users were asked to use the system to listen to music of their choice. We recorded the context the users were in and the songs played in that context. This data was then used for offline evaluation. It was divided into a training set and an evaluation set and the system was trained on the first set. The context readings from the evaluation set were then passed onto the system which output the recommended song for that context. This was compared to the actual song heard and if the two songs are similar[6] we marked the recommendation as correct. In the event of an incorrect recommendation, the system made one more attempt to suggest another song (simulating the user skipping the song). The system's accuracy is then calculated as the percentage of correct recommendations.
For the three participants, we ran the above experiment ten times and calculated the average accuracy. We compared the performance of three systems — a random generator, Adaptive Jukebox, and our system without a context server. The results are shown in table 1. As can be seen our system performs better than random for all three participants and for two participants we achieve a slightly better accuracy result when taking context into account.

	Participant A	*Participant B*	*Participant C*
Adaptive Jukebox	43.45%	21.03%	39.08%
Without Context	39.84%	22.22%	37.30%
Random	19.76%	17.66%	32.13%

Table 1: Evaluation Results

We also plan to run more experiments on the data collected in order to determine whether all context types are important or whether we can remove some sources of context. In the meantime the participants are also using the system in playback mode, where the system recommends songs itself. Evaluation results will be presented in our thesis [Riz07].

6 Conclusion

The low number of participants in our evaluation experiment prevents us from stating any strong claims about our system. However our experiment indicates that incorporating context-sensitivity and using a user model to store musical

[6] In order to check if two songs are similar, we use the user defined similarity between the songs' artists. The user defined similarity is high if the training and evaluation sets contain songs by these artists that are close to each other in the playlists.

preferences in various contexts could help playlist generators make better recommendations. Future work should include better evaluation experiments (e.g. effect of facial expressions) with more participants. Other enhancements might include the incorporation of more features in the song database (such as timbre and instrument detection) and more sources of context (such as physiological inputs for emotion recognition).

References

[OK06] Oliver, N. and Kreger-Stickles, L.: PAPA: Physiology and Purpose-Aware Automatic Playlist Generation. Proc of the 7th International Conference on Music Information Retrieval (2006) 250–253

[PRC00] Pachet, F., Roy, P. and Cazaly, D. A Combinatorial Approach to Content-Based Music Selection. IEEE MultiMedia 7(2) (2000) 44–51

[Riz07] Rizzo, M. Adaptive Jukebox. MSc. Thesis, Department of Computer Science and AI, University Of Malta, Malta. (2007)

[AT00] Alghoniemy, M. and Tewfik, A. H. Personalized Music Distribution. Proc. IEEE International Conference on Acoustics Speech and Signal Processing, ICASSP '00 (2000) 2433–2436

[Log02] Logan, B. Content-based playlist generation: Exploratory Experiments. Proc of the 3rd International Conference on Music Information Retrieval (2002) 295–296

[AP02] Aucouturier, J-J. and Pachet, F. Scaling Up Music Playlist Generation. Proc IEEE Intl Conf on Multimedia Expo (2002) 105–108

[HPD98] Healey, J., Picard, R. and Dabek, F. A New Affect-Perceiving Interface and Its Application to Personalized Music Selection. Proc of the 1998 Workshop on Perceptual User Interfaces (1998)

[ET06] Elliott, G. T. and Tomlinson, B. PersonalSoundtrack: context-aware playlists that adapt to user pace. CHI '06: CHI '06 extended abstracts on Human factors in computing systems (2006) 736–741

[PE02] Pauws, S. and Eggen, B. PATS: Realization and User Evaluation of an Automatic Playlist Generator. Proc of the 3rd International Conference on Music Information Retrieval (2002) 222–230

[AH06] Andric, A. and Haus, G. Automatic playlist generation based on tracking user's listening habits. Multimedia Tools and Applications 29 (2) (2006) 127–151

[FHM01] Field, A., Hartel, P. and Mooij, W. Personal DJ, an Architecture for Personalised Content Delivery. Proc of the 10th international Conference on World Wide Web (2001) 1–7

[PPW05] Pampalk, E., Pohle, T. and Widmer, G. Dynamic Playlist Generation based on skipping behaviour. Proc of the 6th International Conference on Music Information Retrieval (2005) 634–637

[RC03] Ranganathan, A. and Campbell, R. H. An infrastructure for context-awareness based on first order logic. Personal and Ubiquitous Computing 7 (6) (2003) 1617–4909

[HA02] Huron, D., Aarden, B. Cognitive Issues and Approaches in Music Information Retrieval. Commissioned by S. Downie and D. Byrd for a defunct book project entitled Music Information Retrieval (2002)

[LO03] Li, T. and Ogihara, M. Detecting Emotion in Music. Proc of the 4th International Conference on Music Information Retrieval (2003)

[PR00] Pantic, M. and Rothkrantz, L. J. M. Expert System for Automatic Analysis of Facial Expressions. Image and Vision Computing 18 (11) (2000) 881–905

[IC05] Inanoglu, Z. and Caneel, R. Emotive alert: HMM-based emotion detection in voicemail messages. Proc of the 10th international conference on Intelligent User Interfaces (2005) 251–253

[PVH01] Picard R. W., Vyzas, E. and Healey, J. Toward Machine Emotional Intelligence: Analysis of Affective Physiological State IEEE Transactions on Pattern Analysis and Machine Intelligence 23(10) (2001) 1175–1191

[ZGDG03] Zimmermann, P., Guttormsen, S., Danuser, B. and Gomez, P. Affective Computing — A Rationale for Measuring Mood with Mouse and Keyboard. International journal of occupational safety and ergonomics 9 (4) (2003) 539–551

[Jam01] Jameson, A. Systems That Adapt to Their Users: Description of an IJCAI 01 tutorial. Tutorial given at the International Joint Conference on Artificial Intelligence (2001)

[HMI03] Hoashi, K., Matsumoto, K., Inoue, N. Personalization of user profiles for content-based music retrieval based on relevance feedback Proc of the eleventh ACM international conference on Multimedia (2003) 110–119

[Rol01] Rolland, P-Y. Adaptive User Modelling in a Content-based Music Retrieval System. Proc of the 2nd International Symposium on Music Information Retrieval (2001)

[Sch06] Schedl, M. The CoMIRVA Toolkit for Visualizing Music-Related Data. Technical Report. Department of Computational Perception, Johannes Kepler University Linz (2006)

[Dix06] Dixon, S. MIREX 2006 Audio Beat Tracking Evaluation: BeatRoot (2006).

[OSTS06] Oliver, N., Smith G., Thakkar, C. and Surendran, A. C. SWISH: semantic analysis of window titles and switching history. Proc of the 11th international conference on Intelligent User Interfaces (2006) 194–201

[Mae05] Maehr, W. Estimation of the Users Emotional State by Mouse Motions. Diploma Thesis, FH Vorarlberg and Gothenburg University, Dornbirn, Austria. (2005)

[CC06] Cristinacce D. and Cootes, T. Feature Detection and Tracking with Constrained Local Models. Proc of 17th BMVA British Machine Vision Conference (2006) 929–938