Soundtrack for Life - the Context-Aware Music Player

Gregor Bajt
University of Ljubljana, Faculty of Computer and Information Science
Vecna pot 113
SI-1000 Ljubljana, Slovenia
gb3764@student.uni-lj.si

Jovan Toroman
University of Ljubljana, Faculty of Computer and Information Science
Vecna pot 113
SI-1000 Ljubljana, Slovenia
jt3485@student.uni-lj.si

ABSTRACT

As the number of available music grows, the need for music selection grow with it. Classifying by conventional music tags becomes ever more time consuming. A recent approach to musical classification is classifying by context. This paper presents such an approach in the form of a context-aware music player. It uses activity detection for context information together with acoustic features calculated from songs to build context-based playlists. A naive Bayes classifier is used, trained with feedback data gathered from 7 volunteers to make a general model. The model is then shipped with the app and retrained on the fly with further feedback. Our validation results show better accuracy than a model based on random suggestions.

1. INTRODUCTION

With an ever-increasing collection of music to choose from, it might be hard to choose the right song for every situation. The need to make an automatic selection that fits our listening preferences seems more and more apparent. These preferences are decided by the context we find ourselves in at the moment of listening. The context might be trying to relax, commuting to work, exercising, etc. Music, however, is usually not labeled with a specific context, but rather by its title and artist name. These labels are of little use if we do not know how the song sounds like. Genre is a music label that gives us an idea of what a song might sound like, but it is too broad of a descriptor to be of any practical use. Classical music is not necessarily calm, metal music is not always loud etc. We can listen to prebuilt playlists for specific contexts, but these are usually not tailored to our specific tastes with the playlist having bias of the playlist maker. We can construct a personal playlist, but this means updating it every time we add a new song to our collection.

We see that there is a need for a system that automatically classifies songs based on measurable song features. Bergstra et al. (2006) made an algorithm that successfully predicts musical genre and artist using extracted acoustic features such as zero-crossing rate (ZCR) and spectral centroid, among others [1]. Elbir et al. (2018) showed that music genre classification and recommendation can be achieved with machine learning techniques [2]. Magara et al. (2016) propose a context-aware music recommendation system [3]. However, the system does not make recommendations based on acoustic features. Wang et al. (2012) present a model for music recommendations based on activity classification and music content analysis [5]. Their solution, however, relies on internet connectivity for acoustic feature calculation.

We describe and evaluate a novel design for an offline context-aware music player that uses acoustic features to build its knowledge graph. We gather feedback data and use it to train a naive Bayes classifier to make a general model, which is shipped with the app and retrained to make a personalized model. For this purpose we calculate 8 different acoustic features and differentiate between 5 different contexts. The end result is a user-friendly, fun-to-use app which makes it easier for users to enjoy their favorite activities while listening to their favorite songs.

2. RELATED WORK

Magara et al. (2016) propose a context-aware music recommendation system called MPlist. It uses GPS, Wi-Fi, time logs, light sensors, accelerometer and gyroscope data to infer context and activities such as indoor, outdoor, walking, driving in bedroom and in living room. The system learns the user's music preference given the context, previous users listening and other users listening to recommend relevant music genre. The system then builds a playlist with songs labeled with the recommended genre. Our solution predicts if a user would like a specific song in a specific context and then adds the song to the context-based playlist.

Wang et al. (2012) present a probabilistic model for music recommendation that combines automated activity classification with automated music content analysis. They use accelerometer data, hour of the day and noise level recorded with a microphone to infer the following activities: running, working, sleeping, walking, shopping and studying. They calculate 8 different acoustic features on a server to prevent draining the phone battery. A naive Bayes classifier is used to predict if a song belongs to a context-based playlist based on acoustic features and context data. They incrementally train the model with implicit feedback; if the user listens to a song in full, the probability that the song belongs to the current context playlist increases (and vice-versa). A new dataset was constructed from existing playlists for model training. Our solution is similar in many ways, differing mainly in the fact that we calculate acoustic features on the device, thus abolishing the need for internet connectivity. We have also gathered our own data for model training, insuring that the data is in the exact format we need.

3. METHODOLOGY

In this section we describe the building blocks of our application, as well as how those blocks interact amongst each other.

3.1 Application structure

The app has three main functionalities. The first is sensing the user's activity. This is achieved using Google's activity API. We ask for user's current activity every thirty seconds. We use this information to make the next song prediction, as well as a label when gathering user feedback. This activity is displayed next to the song title on the application's main screen. The second functionality is classifying songs by their features. This requires eight selected features of music. We use features that are easier to compute due to low-resource environment of smartphones. These features are Spectral centroid, Compactness, Zero Crossings, Strength Of Strongest Beat, Spectral Variability, Spectral Flux, Spectral Rolloff Point, and Root Mean Square. Lets explore the meaning of these measures. The spectral centroid is a measure used in digital signal processing to characterise a spectrum. It indicates where the "center of mass" of the spectrum is located. Perceptually, it has a robust connection with the impression of "brightness" of a sound. Spectral variability represents the amount of change in song's frequency (does it alternate between low and high frequencies, or is it relatively monotonous). Spectral flux is a measure of how quickly the power spectrum of a signal is changing, calculated by comparing the power spectrum for one frame against the power spectrum from the previous frame. Spectral Rolloff Point is a measure of the amount of the right-skewedness of the power spectrum. In other words, it is the fraction of bins in the power spectrum at which 85% of the power is at lower frequencies. The other features are commonly known or intuitive, so we will not explain their meaning here. These features' extraction is implemented by using jAudio library [4], which we additionally modify to enable it to work in Android environment. After classification, as a part of the third functionality, songs are played automatically once the user engages the "Play!" button. Songs are implicitly organized into playlists for different activities, and random songs from those playlists are played to the user, implicitly providing feedback to our app. The user will either approve or disapprove the track selection and based on user's interaction with the app we infer feedback. This feedback is added to our model data, which is used to train our machine learning system at each app startup. This improves suggested songs for a specific user over time.

3.2 Tehnological stack

We develop our application on Android platform, using AndroidStudio IDE. Our application supports older versions of Android as well (minimal api is 19). Besides the core Android libraries, we make use of: jAudio[1], okhttp[2], firebase[3], and Java Naive Bayes Classifier[4]. We use jAudio for extracting song features. We use okhttp to make HTTP requests for backing up the collected user data in case our primary data collection system malfuntions. As the primary data collection system we use Firebase. This is an easy-to-use and reliable platform from Google. It allows for exporting data in JSON format, which is suitable for input of our machine learning model. We used Python for initially evaluating our model, but we later decided to use a Java implementation for the sake of performance and easier integration.

4. EVALUATION

At the end of each iteration we evaluate the system. We use A/B testing, comparing cases of random song selection and that of songs suggested by our model. In the first half of our testing, subjects are given random song recommendations, and during the other half of the testing subjects use our machine learning model. Once we have the results, we compare them to our current implementation and previous implementations to see if the added feature detection improves track selection accuracy.

4.1 Initial evaluation and data collection

At the very beginning we collect data from seven volunteers on their music preferences. We keep track of their activity, so that we can match specific song features with a certain activity. Besides data about song features and activity, we also collect location and time of the day, which later turns out to be unusable, due to

[1]https://github.com/dmcennis/jAudioGIT
[2]https://github.com/square/okhttp
[3]https://firebase.google.com/
[4]https://github.com/ptnplanet/Java-Naive-Bayes-Classifier

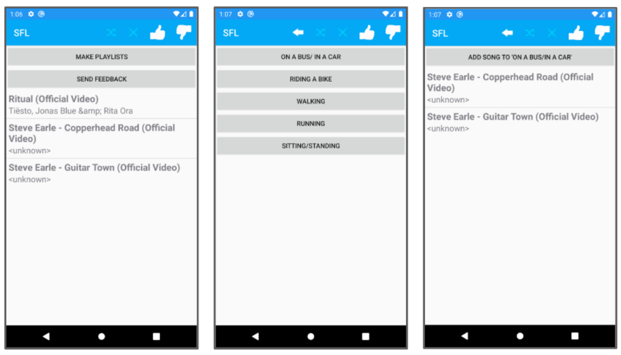

1:06
SFL
MAKE PLAYLISTS
SEND FEEDBACK
Ritual (Official Video)
Tiësto, Jonas Blue &amp; Rita Ora
Steve Earle - Copperhead Road (Official Video)
<unknown>
Steve Earle - Guitar Town (Official Video)
<unknown>
1:07
SFL
ON A BUS/ IN A CAR
RIDING A BIKE
WALKING
RUNNING
SITTING/STANDING
1:07
SFL
ADD SONG TO 'ON A BUS/IN A CAR'
Steve Earle - Copperhead Road (Official Video)
<unknown>
Steve Earle - Guitar Town (Official Video)
<unknown>

Figure 1: Application UI during data collection phase

discrepancies in data. We collect the data in two different ways. First is that users listen to songs on their mobile devices and we collect implicit and explicit feedback from them. Implicit feedback is collected by following a simple rule: if a user listens to half of the song or more, then we record positive feedback. Otherwise, we record negative feedback. Explicit feedback is gathered through "thumbs up" and "thumbs down" buttons in the application user interface (UI). Second way of collecting data is scenario playlists. We give users options to create playlists corresponding to each activity and add the songs they deem appropriate to those playlists. Each of those songs has a positive user feedback. This allows us to collect more data, because many of the users are not able to engage in all the different activities that we want to make predictions for. This phase of evaluation lasted for two weeks. We had seven volunteers which collected data on around seven hundred songs. Some of this data was later deemed as unusable due to discrepancies, so we had around five hundred usable records. This data is later used for calculating accuracy for the random case, and more importantly to train our model. The application was modified for this purpose and the UI can be observed in Figure 1.

4.2 Model evaluation

After collecting data, we move on to model evaluation. We use collected data to train a model to predict if a user will like a certain song while performing a certain activity. The activities that we consider are: walking, running, in vehicle, still and on bicycle. First we filter data to remove entries where song features are inconsistent. Then we split the data based on user feedback to make it easier to evaluate and train our model. We feed train data to the model and evaluate it on test data. When selecting our machine learning model, we evaluate several alternatives. Results for these approaches can be seen in Table 1.

We decide to use Naive Bayes model, because it is

Method	Average accuracy	Average precision	Average recall
NB	0.629	0.652	0.833
KNN	0.617	0.661	0.759
MLP	0.629	0.632	0.925

Table 1: Performance of different methods in predicting song feedback

the simplest amongst alternatives, and it has low processing requirements. This makes it suitable for mobile devices. The model is reevaluated each time a user start up the app. This is so that we can take into account new feedback that is specific for a single user. We add this feedback to our dataset while the user is listening to songs, and then recompute the model at the first app startup. This way the model improves with extended usage.

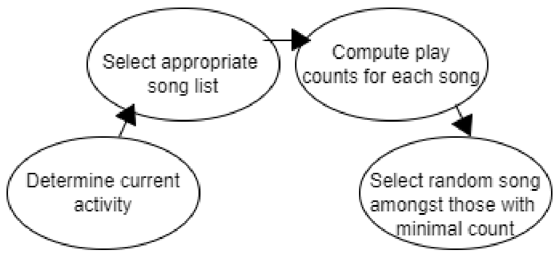

Select appropriate song list
Compute play counts for each song
Determine current activity
Select random song amongst those with minimal count

Figure 2: Next song selection process

At the very first application startup, substantial time is required to compute features for all songs in a user's device. This is necessary so that we can run each song through the model and get a prediction of user feedback. After this first initialization, subsequent startups are much faster because we use cached feature data from our database. After we have results of predictions for each song, we group songs into lists and match them to activities. We match each song that was predicted to be liked by a user to the corresponding activity. Then we use those lists to choose which song to play next. We also implement a system that keeps track of each song play count, so that the same song is not played again until all others have been played at least once. The song selection process goes as depicted in Figure 2. This process can be observed through live data about next song, which is available in the app. The data about the suggestion changes when the current activity changes.

Our model performs reasonably well compared to the random case. We compute the accuracy of random case as:

$$\frac{numberOfCorrectSuggestions}{numberOfIncorrectSuggestions} \tag{1}$$

where a correct suggestion is a song that is liked by a user and incorrect suggestion is a song that is disliked

	Random case	Naive Bayes prediction
Total	369	122
Correct	145	78
Precission	0.393	0.639

Table 2: Results of different prediction approaches

by a user.

We compute test accuracy of our model by splitting our user data into an eighty five percent train data, and fifteen percent test data. Then we feed the song features, user feedback, and activity values as model input, and expect predictions on user feedback when model is provided with only song features values and activity. Model outputs predicted user feedback (like or dislike). We compare the predicted feedback with actual feedback from the fifteen percent test data, and compute test accuracy. Results of random suggestions, as well as results by our model can be observed in Table 2. We can see that Naive Bayes approach achieves prediction results out of the range of statistical error on test data. Figure 3 shows the accuracy of each activity for the random suggestions. Due to an unforeseen error, the activity *running* was not correctly detected and as as such was not included into the evaluation. Out of the 369 entries of feedback, the activities *on bicycle* and *in vehicle* account for only 35 and 5 entries, respectively, which is why they appear as outliers. Dismissing the aforementioned activities, the rest of the activities have a similar accuracy of about 45%. Later in this section we take a look at feedback gathered in the second stage of user evaluation, and see how our model's performance compares to that achieved on test data.

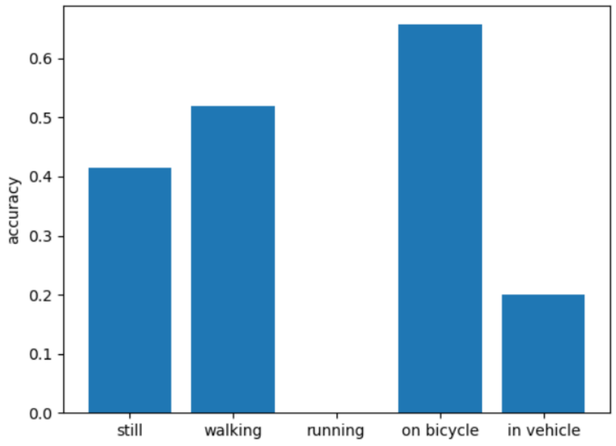
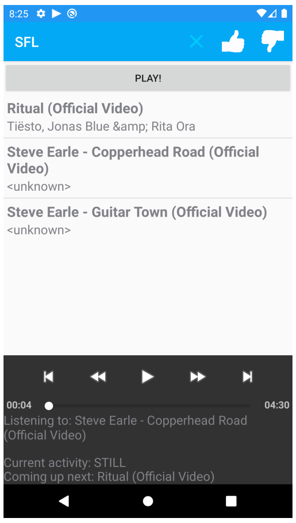


Figure 3: Accuracy by activity for random suggestions

4.3 Application performance

After we setup the model, we move on to real-environment app evaluation. For this phase, only five of the seven users from the previous phase participate in the evaluation. At this phase, app UI is simplified. There are still explicit feedback buttons, as well as a "Play!" button which start the app algorithm. After the algorithm is started, a user can control playback as expected in a music player. If no action is taken by the user, our algorithm plays next suggested song after the previous one is completed. We record implicit and explicit feedback and send it to our server. We then use this data to evaluate our model's real performance. The results are discussed later in this subsection. The app UI in this phase also features real-time activity and next song information. It is interesting to see this info change when the current activity is changed. App UI in phase two can be observed in Figure 4.



Ritual (Official Video)
Tiësto, Jonas Blue &amp; Rita Ora

Steve Earle - Copperhead Road (Official Video)
<unknown>

Steve Earle - Guitar Town (Official Video)
<unknown>

00:04 04:30
Listening to: Steve Earle - Copperhead Road (Official Video)

Current activity: STILL
Coming up next: Ritual (Official Video)

Figure 4: Application UI during real-environment app evaluation phase

In the second phase we gathered 275 valid entries of feedback. An entry counts as valid if the detected activity of the entry is valid (not 'unknown'). Figure 5 shows accuracy for each activity. The average accuracy of all activities is 62.5%. Activities *still*, *walking*, and *running* seem to have accuracy which is close to average. As we move from calmer activities towards more energetic ones (from *still* to *running*), the accuracy seems to worsen by about 10%, indicating that the model is better at recommending songs for low intensity activities. This might be the result of having more data for the calmer activities.

Activities *on bicycle* and *in vehicle*, however, seem to be outliers. This is likely due to the fact that the number of entries for these activities are 9 and 4, respectively. We believe that these results are not repre-

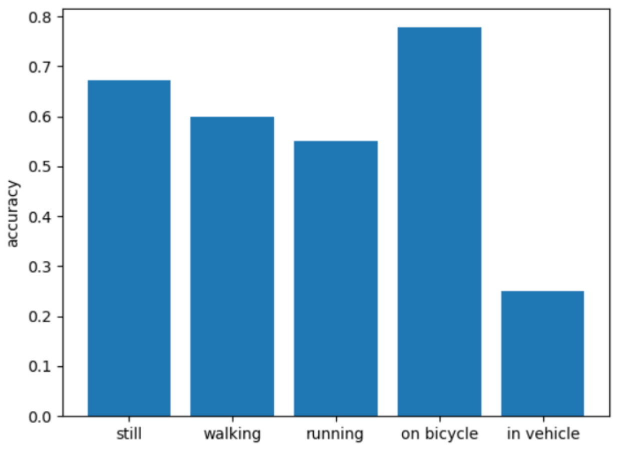


Figure 5: Accuracy by activity using our model

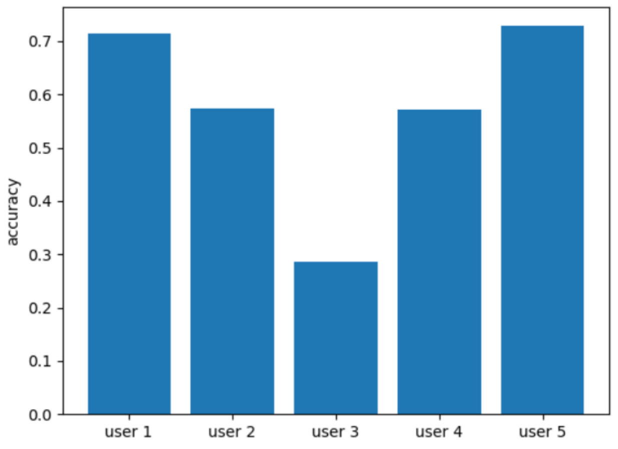


Figure 6: Accuracy by user using our model

sentative of the true performance of the model because of the low entry count. Gathering more entries for both activities would result in a more true representation of accuracy.

Figure 6 shows accuracy for each user. Users 3 and 4 have supplied us with 21 and 7 entries, respectively. We deem the number of entries for these users to be too low to give an accurate representation of accuracy. The rest of the users show an accuracy of above 70%, with the exception of user 2, who has an accuracy of just below 60%. If we dismiss the outliers for the low entry count, all users seem to have a consistent accuracy around the average.

5. CONCLUSION

We implemented an offline music player that uses a combination of contextual information and acoustic features to build context-based playlists. We gathered feedback data to train a machine learning model to build a general model. The model is then retrained with further feedback. We conclude that this application achieves accuracy that is beyond the rate of statistical error. With this in mind, we look at aspects which can be improved and emphasize the things that already perform well and do not have to be changed significantly.

We could improve energy efficiency by using Google's activity transition api instead of the regular activity api. This would decrease the impact our app has on a user's energy budget, which has become of increasing concern in the last couple of years. Another thing we can improve is feature calculation performance. We had problems with file reading due to Android system's restrictive file write policies. This can be improved with further investigation into Android's file write permissions api.

An obvious way to improve our model would be to collect more data from all users. This would include gathering feedback from all users after they install our app. The reason we currently do not employ this approach is due to our conservative privacy policies. Another improvement would be experimenting with different machine learning approaches, e.g. neural networks. These approaches, although requiring more data can make a better model of the real system than Naive Bayes, which uses only a-priori and a-posteriori probabilities.

Finally, we leave open possibility of using more complex acoustic features such as MFCC, Zernika Moments, and others, once our system performs well enough that we can afford such overhead. We could also gather and use more contexual information. We currently only use detected activities for context. The app does track location and time but these are not used because of data discrepancies in the training data. Collecting a better dataset would allow us to improve context recognition.

The complete source code of our app can be found in our GitHub repository [5].

CONTRIBUTIONS

Jovan Toroman: base music player, ML model implementation and evaluation, basic feature extractor, secondary feedback gathering system
Gregor Bajt: activity and location detection, scenario playlist feedback system, storing feedback to server
Both: volunteer recruitment, writing paper, UI

6. REFERENCES

[1] Bergstra, J., Casagrande, N., Erhan, D., Eck, D., and Kégl, B. Aggregate features and adaboost for music classification. *Machine Learning 65* (12 2006), 473–484.

[2] Elbir, A., Bilal Çam, H., Emre Iyican, M., Öztürk, B., and Aydin, N. Music genre

[5] https://github.com/JovanToroman/SoundtrackForLife

classification and recommendation by using machine learning techniques. In *2018 Innovations in Intelligent Systems and Applications Conference (ASYU)* (Oct 2018), pp. 1–5.

[3] Magara, M. B., Ojo, S., Ngwira, S., and Zuva, T. Mplist: Context aware music playlist. In *2016 IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech)* (Aug 2016), pp. 309–316.

[4] McEnnis, D., McKay, C., Fujinaga, I., and Depalle., P. jaudio: A feature extraction library. *Proceedings of the International Conference on Music Information Retrieval* (2005), 600–3.

[5] Wang, X., Rosenblum, D., and Wang, Y. Context-aware mobile music recommendation for daily activities. In *Proceedings of the 20th ACM International Conference on Multimedia* (New York, NY, USA, 2012), MM '12, Association for Computing Machinery, p. 99–108.