Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Osti: Personalised Context-Aware Music Recommendations in Real-Time

Author:
Jack Morrison

Supervisor:
Dr William Knottenbelt

Second Marker:
Professor Alessio R. Lomuscio

June 14, 2021

Abstract

Existing music recommendation techniques currently available for workouts either are generic, and not curated to a specific user's music taste, or don't take into account a user's workout history to produce targeted recommendations. We introduce a platform, Osti, which tackles both of these issues. It produces recommendations for any possible workout type a user can record, using their complete listening history from streaming services, helping them to achieve their ideal workout based on learned or user-specified targets.

This resulted in the creation of a personalised music recommendation system, producing recommendations based on a user's workout context, both asynchronously in the form of playlists, and in real-time. The application consists of multiple microservices for: data collection, recommendation generation and playlist creation, as well as a web interface for viewing and updating this data. It also resulted in a WatchOS application for user vital collection to improve music recommendations in real-time.

Current empirical results show that Osti outperforms almost all other pre-generated workout playlists in keeping a user close to their target workout vitals, and has similar or better success to user-curated playlists in most workout types. It also is shown to improve on this performance using real-time recommendations to adapt the music playing for constant-exertion workouts such as running.

Acknowledgements

I'd like to thank my supervisor and tutor, Dr William Knottenbelt, for his useful feedback and guidance to help shape this project.

To my parents and sister, thank you for always encouraging me to reach my potential, I wouldn't be where I am today without you and your unwavering support.

Finally to my friends, thank you for your constant support, laughter and late nights working in labs over the last four years, it's been a blast.

Contents

1 Introduction **5**

1.1 Motivation . . . 5

1.2 Objectives . . . 5

1.3 Contributions . . . 6

2 Background **7**

2.1 Recommendation Systems . . . 7

2.1.1 Utility Matrix of Preferences . . . 7

2.1.2 Content-Based Systems . . . 8

2.1.3 Collaborative Filtering . . . 8

2.1.4 Hybrid Solutions . . . 10

2.1.5 Similarity Algorithms . . . 11

2.2 Similar Existing Systems . . . 12

2.2.1 Context-aware Mobile Music Recommendation for Daily Activities . . . 12

2.2.2 Sequence-based Context-aware Music Recommendation . . . 14

2.2.3 MusicalHeart: A Hearty Way of Listening to Music . . . 15

2.2.4 Spotify's Soundtrack Your Workout . . . 17

2.3 Background Summary . . . 19

2.3.1 Common Themes . . . 19

2.3.2 Common Problems . . . 19

3 Implementation **21**

3.1 Overview . . . 21

3.2 Data Fetching Pipeline . . . 22

3.2.1 Connecting to APIs . . . 22

3.2.2 Listening History Collection . . . 23

3.2.3 Workout History Collection . . . 23

3.3 Music Recommendation Engine . . . 23

3.3.1 Version 1: Popularity Based . . . 25
3.3.2 Version 2: Content-Based Recommendations . . . 25
3.3.3 Version 3: Hybrid Collaborative Filtering and Content-Based Recommendations . . . 26
3.3.4 Version 4: Reintroducing Popularity Metrics and Categorical Features to the Hybrid System . . . 27
3.3.5 Version 5: Adding a User Feedback Loop . . . 28
3.4 Playlist Generation . . . 29
3.4.1 User Workout Targets . . . 29
3.4.2 Workout Profiles . . . 30
3.4.3 Workout Deltas . . . 30
3.4.4 Song Deltas – Coarse Grained . . . 32
3.4.5 Song Deltas – Fine Grained . . . 32
3.4.6 Using workout deltas, song deltas and recommendations to generate a playlist 33
3.5 Real-time Track Recommendations . . . 34
3.5.1 Initial Data Collection . . . 34
3.5.2 Initial Tracklist . . . 35
3.5.3 Calculating Ideal Tracks . . . 36
3.5.4 Switching Tracks . . . 37
3.6 User Interface . . . 37
3.6.1 Initial Setup . . . 38
3.6.2 Recommendations and User Feedback . . . 38
3.6.3 User Workout Targets . . . 39
3.6.4 Generated Playlists . . . 39
3.6.5 Missing Songs . . . 39

4 Evaluation **41**
4.1 Data Visualisation . . . 41
4.1.1 Feature Analysis . . . 41
4.1.2 Known vs Unknown Recommended Tracks . . . 43
4.2 User Testing . . . 44
4.2.1 Playlist Generation . . . 45
4.2.2 Real-time Recommendations . . . 51

5 Ethical Considerations **54**
5.1 Human Participation . . . 54

5.2 Data Protection . . . 54

6 Discussion, Future Work and Conclusion **55**

6.1 Discussion . . . 55

6.2 Future Work . . . 55

6.2.1 Customisable Workout Target Vectors . . . 56

6.2.2 Further Research into Real-Time Recommendations for Non-Constant Exertion Workouts . . . 56

6.2.3 Temporal Recommendations . . . 56

6.2.4 Lyric Comprehension & Music Tagging . . . 56

6.3 Conclusion . . . 57

A Database Schema **58**

B User Testing Results: Playlist Generation **59**

B.1 Osti Playlist . . . 60

B.1.1 Running . . . 60

B.1.2 Walking . . . 61

B.1.3 Strength Training . . . 62

B.2 Spotify's Soundtrack Your Workout Playlist . . . 63

B.2.1 Running . . . 63

B.2.2 Walking . . . 64

B.2.3 Strength Training . . . 65

B.3 User Hand-Curated Playlist . . . 66

B.3.1 Running . . . 66

B.3.2 Walking . . . 67

B.3.3 Strength Training . . . 68

B.4 Generic Workout Playlist . . . 69

B.4.1 Running: 'Fun Run' Playlist . . . 69

B.4.2 Walking: 'Walking Music' Playlist . . . 70

B.4.3 Strength Training: 'Workout' Playlist . . . 71

C User Testing Results: Real-Time Recommendations **72**

C.1 Running . . . 73

C.2 Strength Training . . . 74

Chapter 1

Introduction

1.1 Motivation

With the growth of digital media, the amount of music that's readily available is dramatically increasing each day, and so it's becoming an ever-growing challenge for listeners to filter through and find the songs they prefer most. In 2017, Spotify confirmed that they add around 20,000 songs a day to their catalogue, moreover they also stated in 2020 that this had grown to nearly 40,000 songs daily [1], showing how rapidly the streaming industry is growing.

It's also been well researched that music can help improve a person's performance while doing many activities, such as exercising. One report in the Journal of Sports Sciences [2] stated that there are many different benefits which listening to music while doing exercise brings, for example diversion of attention [3], triggering or regulation of specific emotions and moods [4] and encouragement of rhythmic movement [5], which all lead to increased performance. However, with the overwhelming number of songs available, it can be difficult to select the optimal music to produce the best results.

Currently, the most common way to get music for a workout is to personally curate a playlist for every possible workout type. However, this takes time to thinks of tracks and order them for the workout, and these playlist are also usually limited to tracks the user knows - they don't allow new music to be discovered that may be even better suited for a specific workout context. This is very slow and wastes a lot of time, and therefore is the perfect candidate to be sped up by automation.

This is where Music Recommendation Systems come in. They're usually used to create playlists for users. These are most commonly one of two types - personalised playlists, such as Spotify's 'Made For You', and activity- or context-specific ones, such as a pre-made 'Home Workout' playlist. However, these both lead to problems; personalised playlists may have tracks inappropriate for the current context, e.g. songs which are too slow for running, and curated playlists don't take a user's music taste into account. This therefore leaves a gap in the market, since it's known that people have context-based music preferences [6], but there aren't systems which take advantage of this.

However, even if these playlists were generated perfectly for the user's ideal workout, their actual workout may not go according to plan. Therefore, being able to adapt these recommendations in real-time, to keep a user on track to hit their workout targets without over-exerting themselves, would allow for a more useful system.

1.2 Objectives

The main objective of this project is to create a system that automatically generates user-specific playlists for different workout types, and be able to adapt these recommendations in real time to keep a user on track to reach their workout targets. This tackles the following key problem areas:

- **Context-Aware Recommendations**. People listen to different types of music whilst they do different activities. Suggestions should be tailored to a specific workout context, e.g. running, weightlifting or Pilates.
- **Personalised Suggestions**. People have their own music taste, which carries through to when doing different activities. Therefore, suggestions should take into account a user's personal likes and dislikes through the user of their listening history.
- **Automation**. Currently, to create a playlist like this requires lots of manual curation. The system should generate these playlists automatically, with minimal user interaction, gathering the data needed from existing sources such as listening history, and workout data from personal devices such as smart watches.
- **Real-time Updates**. Workouts don't always go to plan, and therefore the system should be able to adjust these recommendations in real-time to constantly keep the users vitals at their ideal level for them to reach their targets.

1.3 Contributions

The project has produced a end-to-end system, Osti, which automatically retrieves users workout and listening data (Section 3.2) and uses this to generate recommendations of tracks which will help them in achieving their target workout, with respect to target vital values such as heart rate and distance covered (Section 3.3). These recommendations come in the form of automatically generated playlists (Section 3.4), as well as an Apple Watch application to provide real-time updates to these recommendations, adapting to changes in the workout (Section 3.5). The system can managed through the online web application located at `https://osti.uk` (Section 3.6).

Osti is a system that uses not just musical features such as tempo to recommend tracks, but also the historical impact of the track on workout performance. This impact is shared across everyone who uses the application using collaborative filtering, meaning the system learns and shares knowledge between users, so as more users enter the system, the amount of data powering recommendations also increases.

Chapter 2

Background

2.1 Recommendation Systems

Recommendation systems are applications which, when given a dataset containing multiple items, will predict which items a particular user would most prefer. An example is the video streaming platform Netflix, which uses recommendation systems to predict which videos a user is most likely to want to watch. This is done using their watch history [7]. There are two main groups of technologies used by recommendation systems, which are content-based systems and collaborative filtering.

2.1.1 Utility Matrix of Preferences

One model which is very useful for explaining recommendation systems is the utility matrix of preferences, as presented in the book *Mining of Massive Datasets* by Leskovec et al. [7]. It is used to represent the preferences between users and items in the dataset. This is represented in the form of a matrix, where each pair of user and item contains a value which represents the preference of a user for that item. One example of this could be an integer rating of 1 to 5. The majority of the entries in this matrix are unknown, meaning the system has no explicit preference value for a user-item pair.

	Shake It Off *Taylor Swift*	Blank Space *Taylor Swift*	Style *Taylor Swift*	Baby *Justin Bieber*	Don't Start Now *Dua Lipa*	Levitating *Dua Lipa*	Physical *Dua Lipa*
A	5	5	5			2	
B				5	5		4
C	1				5		4
D		3		2			

Table 2.1: A utility matrix representing song ratings on a 1-5 scale

As seen in Table 2.1, the blank spaces represent unrated songs. As the size of the dataset grows, the sparseness of the utility matrix will also increase. The aim of the system is to fill in the blanks of this matrix, the two ways to do this being to use content-based systems and collaborative filtering. The former involves using features of the songs, such as tempo and artist, so for example, since user C likes the songs Don't Start Now and Physical, they're very likely to also like Levitating, since they all share the same artist. The alternative method is collaborative filtering, so for example, since users B and C both like the songs Don't Start now and Physical, and B likes the song Baby, C will probably like the song Baby also.

We don't need to fill in all of the blanks, since this can be very computationally expensive. We only need to find the values which will have a high value, since these will be recommended to the user.

The given values in a utility matrix can be hard to collect. There are two main methods used to get these values. The first is by getting users to explicitly rate items, in our case songs. This

can be effective since the rating you get from a user is accurate to their preference, however, it's limited in reality since users aren't always willing to rate items. Therefore, the alternate method is to infer preference from a user's behaviour. For example, if a user listens to a song more than once, it can be assumed that they like it.

2.1.2 Content-Based Systems

As stated earlier, a recommendation systems consists of a set of users and a set of items which can potentially be recommended to the users. For each of these potential items, a profile is created, which is a set of features that represent this item. [7] For example, in a music recommendation system, these items to be recommended would be songs, and some features could include artist, genre, tempo and key. These features are all usually given with the song, however features could also be learned data, such as 'danceability', which are predicted using machine learning models.

When representing an item, there are two main types of features which need to be represented: discrete and numerical features. The former are things like artist and genre, which can be picked from a finite set, and they're either part of the item profile or not. For example, if we chose artist as a feature of a song, then feature this can be represented by a vector, where each component is every possible artist, and the value is 1 if the artist performs in the song, and 0 if not. Numerical features cannot be represented by a boolean vector, and so are things like tempo, where it's important that if two songs are close in tempo, they're more likely to be similar than two songs which vary in tempo by a greater value.

Once we have an item's profile, we need to be able to measure how similar two items are. There are many different ways of doing this, and these are compared in Section 2.1.5. One thing to take note of is that numerical features need to be scaled so that they do not dominate the measurement.

The other key aspect of a recommendation system is being able to represent a particular user's preferences through a user profile. These profiles have to have the same components as the item's profiles, so that they can be easily compared. Each of these features in the user profile can be represented similarly as discrete or numerical features, where the discrete value of 0 and 1 could represent if a user has listened to a particular artist, or a numerical value could give their rating of that artist, for example based on how much they have listened to them.

An example for a user profile, based on the utility matrix in Table 2.1, could be that user A has listened to 4 songs, 3 by Taylor Swift and 1 by Dua Lipa. Therefore, 75% of the songs they've listened to are by Taylor Swift, and 25% are by Dua Lipa, so the user profile will have 0.75 and 0.25 respectively for these two artist's components. However, to make this more accurate, we could take into account their rankings for the songs they've listened to. We can calculate the average rating value for A from all rankings, which is 4.25, and subtract this from all ratings. This means any ratings below average are negative, and any above average are positive, which is useful in recommendation. Therefore, the new value for Taylor Swift would be the average of 5-4.25, 5-4.25 and 5-4.25, which is 0.75. On the other hand, their ranking of Dua Lipa would be like 2-4.25, which is -2.25.

The final stage is to recommend items to a user based on their content. This is done using both the item and user profiles, using a similarity algorithm as when measuring how similar two items are. Some other optimisations can be applied here in order to not have to check every possible match, such as random hyperplanes and LSH, to put items into different buckets, and use these techniques on the users to decide which buckets to look in.

2.1.3 Collaborative Filtering

The other main technology used in recommendation systems is collaborative filtering. This is different to content-based models since it does not take into account any of the features of the data to be recommended. Instead, it uses the similarity in how users rate specific items to determine how alike they are. For example, in a music recommendation system, to recommend songs to a particular user, users who listen to similar music to that user are found, and songs that they like

will be recommended. [7]

The first area we have to think about is how to measure the similarity of two users. This is discussed more in Section 2.1.5.

There are two main types of collaborative filtering; user-based and item-based. User-based collaborative filtering works by taking a particular user A, and using a similarity algorithm to find the set of most similar users U_A. Then, the recommendations can be made based on what songs these similar users like. For example, say we want to estimate the rating of song X, which user A has not rated, but members of their similarity set U_A have. First, you normalise you averaging the rating of the users in U_A who rated song X, by subtracting their average rating. This accounts for any users who give high or low ratings. Then, we average the normalised ratings for all users in U_A who rated X. Then, this normalised average is added to user A's average rating for all items. This then gives a predicted rating for song X by user A.

On the other hand, item-based collaborative filtering works by comparing the similarity of items, in this case songs, rather than users. This is done by finding the set of most similar items to song X, S_A. These rankings are then again normalised, and then the average ranking of these songs for user X is calculated.

In both of these methods, at least the values with a high predicted ranking need to be estimated, in order for items to be recommended for the user. The advantages of user-based collaborative filtering are that we only need to find the set of similar users once, and then we can use it to fill in the remainder of the utility matrix. On the other hand, with item-based collaborative filtering, we need to need to find the similar items for all items before we can fill in the gaps for a particular user. However, for item-based collaborative filtering, the results are more reliable, since you know both of the items will definitely be similar, and therefore the user will most likely like it, whereas with user-based collaborative filtering, a similar user may have a slightly different taste, which means some songs they like may not match up with the user looking for recommendations, and so there is no guarantee they will like the recommendations.

One advantage of collaborative filtering in general is that each user's preferred items can be precomputed, since the utility matrix changes slowly over time, so it needs to be recomputed infrequently.

However, a disadvantage of collaborative filtering is that the data can be sparse, and therefore it may be hard to find similar users or items. One way to fix this is to cluster items and/or users together. A similarity algorithm can be used to cluster items together, or it could be done on a specific feature. For example, if we clustered the utility matrix in Table 2.1 by artist, we would end up with the following:

	Taylor Swift	Justin Bieber	Dua Lipa
A	5		2
B		2	4
C	1		4.5
D	3	5	

Table 2.2: A utility matrix representing song ratings on a 1-5 scale, clustered by artist

This is calculated by the rating for a user of a cluster being the average rating that they gave to all of the members of the cluster that they rated. This process can be repeated multiple times, for example it could then be done by grouping all artists of the same predominant genre. Users could also be clustered using a similarity algorithm in an equivalent manner.

Once the utility matrix is clustered appropriately so that it is less sparse, then to to find the rating of song X for user A, look up which clusters X and A belong to, and if the entry is non-blank, use this value. If the value is blank, then use a similar method to above where similar clusters are considered.

2.1.4 Hybrid Solutions

Both of the solutions above offer advantages and disadvantages, which can be seen below in Table 2.3.

	Content-Based Systems	**Collaborative Filtering**
Advantages	- Doesn't require data about other users - Can learn specific user interests - Can recommend unknown items	- No item features needed - Allows items with different features to be discovered that will most likely be liked
Disadvantages	- Requires item features of good quality - All recommendations are similar to user's taste, therefore taste isn't expanded	- New items have no data - Data is sparse

Table 2.3: Content-Based Systems vs Collaborative Filtering [8] [9] [10]

Therefore, the way to go about mitigating these disadvantages are to combine the two methods, as is completed in many papers, including one by Su and Chiu [8]. The main problems which are tackled are:

- **New items have no data.** This occurs when no user has rated a particular item. This means that there is no way of predicting if a user will like that item. This is mitigated by using item features.
- **Sparse Data.** This is where the utility matrix is very sparsely populated. The more sparse this matrix is, the harder it is to predict user ratings [8]. This is also mitigated by using item features.
- **Recommendations don't expand taste.** This means that only similar items would be suggested, however there may be patterns not detected by the features of the items which mean people usually like both items. The incorporation of collaborative filtering allows these patterns to be discovered.

The hybrid system essentially uses item-based collaborative filtering to predict user rating, however uses the similarity of music content instead of music rating. This allows the user's music taste to be predicted, even when data is sparse. Su and Chiu's system uses low-level music features, however this could be extended to use other features such as tempo and artist.

Their system is broken down into two phases, offline preprocessing and online prediction. Offline preprocessing is used in in order to accelerate the online prediction stage. Offline preprocessing consists of extracting the features of the music, and then using these to generate the item-to-item similarity matrix, using a similarity algorithm.

Online prediction only occurs when a user is looking for recommendations. This phase uses the pre-computed similarity matrix to figure out all of the users unknown rankings. For each unknown ranking, the similarity matrix is used to figure out the most similar songs to that song quickly, and then the unknown ratings of the target can be computed by item-based collaborative filtering.

A mathematical example from Su and Chiu's paper is as follows. The most relevant item set to target item itm_i for an active user u_z is $U_z = \cup itm_p$, where itm_p are unique items of music. The rating v_i^z of itm_i for user u_z is defined as:

$$v_i^z = \frac{\sum_{itm_p \in U_z} \text{sim}_{i,p} \times v_p^z}{\sum_{itm_p \in U_z} \text{sim}_{i,p}}$$

Where $\text{sim}_{i,p}$ is the result from the pre-computed similarity matrix for items i and p.[8]

2.1.5 Similarity Algorithms

One important part of a recommendation system is deciding how similar two items are. There are many algorithms to do this, each with their own advantages and disadvantages.

2.1.5.1 Jaccard Index

Jaccard Index is used to measure the similarity between two sets. It's translated to vectors which consist of binary values, such as songs which have been listened to and ones which have not. It's calculated by taking the intersection the components of the vectors which are 1s, and dividing by the union of this. [11] It can be written:

$$\text{dist}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Where $|A \cap B|$ represents the number of components the two vectors have set to 1 in common, and $|A \cup B|$ represents the number of components they have set to 1 between them. [12]

The Jaccard Index doesn't take into account how much a user likes a particular item in the utility matrix, just which items they have rated. Therefore, it loses information when there is a metric to how much a user likes a specific item.

2.1.5.2 Cosine Similarity

Cosine Similarity is one of the most frequently referenced similarity algorithms in recommendation systems literature [6–8]. It measures similarity by calculating the cosine of the angle which the two input vectors make in their dot product space. [11]

Given two vectors of features A and B, the cosine similarity between them is calculated using the formula:

$$\text{sim}(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

The resultant value is between 1 and -1, where 1 is a perfect match, and -1 means they're opposites. However, one downside is that it does not take into account the length of the vector.

Another problem is that blank values in the utility matrix are treated as 0s, which means that they are interpreted as a dislike, however this is something which can be improved upon with hybrid solutions, by using song features to predict a users preference towards a song, so there are less blank values [8].

Cosine Distance is a commonly used to reference the inverse of Cosine Similarity, where:

$$\text{Cosine Distance} = 1 - \text{Cosine Similarity}$$

2.1.5.3 TS-SS (Triangle's area Similarity - Sector's area Similarity)

To tackle the drawbacks of cosine distance, namely that the length of the vector is not taken into account, TS-SS was developed. This makes sense when trying to find the similarity between ratings, since songs with higher ratings are what the system is looking for in order to recommend, and therefore should have a higher similarity. However, it may not translate as well to features such as tempo, since a higher tempo does not necessarily mean a higher similarity.

TS-SS works by calculating the area in the triangle between two vectors, and then subtracting a sector which is calculated using the Euclidean distance between the vectors, which is needed for

when two vectors of different lengths have the same angle. This allows it to take into account not just the angle between the vectors, but also the length of the vectors [13].

2.2 Similar Existing Systems

The current landscape of music recommendation systems is pretty wide, and research is ongoing in multiple areas. The two methods of collaborative filtering and content-based methods detailed above are what the majority of new emerging techniques are built upon, and there are many more state-of-the-art approaches which build upon these.

However, one thing with these methods is that they usually take into account lots of data over a long period of time. They basic versions described above are not necessarily designed to take into account short-term fluctuations in what a user wants to listen to. This can change based on the activity being completed, and therefore some other ideas have been proposed to figure out these activities and use them to help influence music recommendation.

In this section we will explore some similar systems in depth, and provide a thorough evaluation of each system.

2.2.1 Context-aware Mobile Music Recommendation for Daily Activities

2.2.1.1 Overview

This was a case study from the National University of Singapore, which looked into using contextual information collected from low-level mobile phone data, along with music context analysis, to recommend music for daily activities [14]. These are things such as working, studying, running and shopping. Their efforts culminated in an Android application.

The paper's main contributions are split into four parts:

- **Automated activity classification:** The paper uses low level data from a mobile phone, specifically time, accelerometer data and microphone data, in order to predict the current activity of the user. The do this by using a probabilistic model to classify the activity into either working, studying, running, sleeping, walking or shopping.
- **Automated music content analysis:** The next major contribution is to classify how appropriate a song is for each of the particular activities listed above. They don't assign one category per song, but use a probability, so that a song can be recommended for multiple activities. This is done using another probabilistic model, which takes into account the user feedback on each song, for example, by how quickly they skip it.
- **Solution to the cold start problem:** One problem most music recommendation systems is the cold start problem, which is when the system has no information on a song or on a user, so they can't accurately recommend songs. This is because most other system use some kind of collaborative filtering, which requires some form of user ratings or manual annotations. However, by using music content analysis, the features of a song can be pre-calculated, and by being able to quickly detect activities, a good starting recommendation can be given to the user, which improves over time.
- **Implementation and evaluation:** The paper implements this all directly on a mobile phone, except the pre-computed music content analysis, which is done on a sever. The paper also gives a very thorough evaluation of the accuracy of the application, and explains how user testing was implemented for continual feedback.

The first two of these contributions are the key to creating the algorithm to suggests songs. The first, also known as *context inference*, uses a feature vector, collected from the mobile phone's time, accelerometer and microphone data, in order to calculate a probability distribution over the range of possible contexts (or activities). It does this using a Naive Bayes method, since this has

a fast prediction and allows fast incremental training.

The second contribution, also known as *music content analysis*, combines both features of music, along with user feedback on how much a user like a certain song, to suggest songs which correlate to a certain context. It uses a method called Autotagger to initialise the model between the music and the context, which is the same for all users. The probability of a user liking a song based on a context is then modelled by a random variable R, which is updated as a user provides feedback on the song.

The two equations for both of these contributions are combined to generate probabilities a user will like each song based on the current context. Finally, a recommendation task is created, which is to maximise the user satisfaction for the context by giving the most appropriate songs.

2.2.1.2 Evaluation

The evaluation of the application is very thorough, analysing the accuracy of the music-context model and the sensor-context model, as well as using lots of user feedback throughout the process.

For the music-context model, it classified the six activities mentioned earlier, and worked especially well for running and sleeping. The paper doesn't elaborate on why this is, however these activities both tend to have a very specific type of music associated with them - sleeping being calm, relaxing music, and running usually high-energy music.

For the sensor-context model, the paper compares the approach of Naive Bayes against 5 other approaches. The results show that out of all of the approaches chosen, the Naive Bayes approach performed the worst. However, this model was chosen since it can give a fast prediction, and allows for incremental training (meaning the sensor context models don't need to be completely retrained each time more data is collected). This was a limitation of the mobile technology available at the time, and therefore the choice of classifier is something which could be improved upon, since more complex classifiers would be able to perform in efficient time on today's devices. Another thing noted is that the 'working' and 'studying' categories were not well distinguished by any of the classifiers, since the information gathered is usually very similar. This may benefit from other features, such as user age or location, in order to distinguish between them.

The user study undertaken throughout the project used 10 participants who all used the application. These users were also questioned about the need for such an application, to which the results were overwhelmingly encouraging. Since this paper focused on daily activities, and used mobile phone data, whereas most existing papers in the area at the time didn't, it was hard to make a direct comparison to any other systems. Therefore the paper does a comparison between recommending songs randomly, using the 'auto' mode, where the context category was inferred, and the 'manual' mode, where the context category is given by the user. They created a control group and an experimental group, giving the control group random recommendations, and the experimental group data from the application. The users did each detectable activity for around 20 minutes, and then rated the song on a scale of 1 to 5 based on how much they liked it. The results were best for the manual mode of the application, closely followed by the auto mode, both averaging around 4/5. The random recommendations still scored fairly highly, at around 3.5/5. However, the difference between manual and auto mode was very small, and therefore showed that their sensor-context model was fairly accurate.

One problem with this evaluation is that these experiments only took place over a small time frame. Therefore, the authors also did an adaptation evaluation, in order to see how the recommendations changed for two of their users over the course of a week. The results showed a massive increase in accuracy of both context inference and song recommendation, from 87% to 96% and 68% to 93% respectively. This gives confidence to the idea of the application learning about the user over time in order to improve recommendations and more accurately detect activities.

The paper is very obviously limited by the mobile technology available at the time - for example, only using the features of time, accelerometer data and microphone data to train it's sensor-context

model. However, the results still show an increase in recommendation ratings with the application in comparison to randomly generated songs in the user evaluation. The user evaluation also proved that over a more significant amount of time, the recommendations increase in accuracy, and therefore this idea is worth investigating further. With the increase in types of data which are able to be collected from users, for example heart rate and more detailed fitness and sleep tracking, using devices such as smart watches, the accuracy of a sensor context-model is likely to increase significantly.

The main takeaways from this paper are:

- Incremental Training is very useful since it doesn't require all historic data for every user to be stores, which is unsustainable. For sensor data, it just requires one model per user, which can be made more accurate as data is collected, and then this data can be discarded once it has been used to update the model.
- To prevent the cold-start problem, new songs should have their features analysed in order to have data about which contexts they are most appropriate for. For users with no listening history, this is enough to still provide songs, however over time, more accurate suggestions can be calculated. For users with existing listening histories, these can be used to pick appropriate songs initially.

2.2.2 Sequence-based Context-aware Music Recommendation

2.2.2.1 Summary

Another important aspect of music recommendation is the ability to recommend songs which are relevant to a listener at a current point in time. This paper, by Wang et al. [6], does this by inferring a user's music preference and giving recommendations in real time.

One key point about this paper is that is refers to the *context* a user is in, rather than a specific activity they are doing, as the National University of Singapore paper [14] does. This encompasses more than just an activity they're doing, but also could relate to their emotion, the time of day, and their physical surroundings.

The method works by first creating a low-dimensional representation of pieces of music using neural networks. Songs which should be recommended in the same context should have similar representations. then infers and models each user's general and contextual preferences from this. Their general preference is their music taste over the whole history, and contextual preferences are based on what they've been listening to in their last listening session. Finally songs that conform to these preferences are recommended in real time.

One thing to note is that it combines both music features with collaborative filtering, since if a user plays songs together in a similar context, this will influence other users' recommendations.

2.2.2.2 Observations

This paper made three key observations about a user's listening data, using existing works on music recommendations, which are helpful in proving the feasibility of the overall project:

- Every user has a *music taste* - the type of music they like, which can be inferred from their listening history [15].
- For different contexts, users will like different styles of music, so their taste is influenced by the context they are listening in, but is not the same for everyone [16]. An example used is that someone who likes light music and rock music may like the former when resting, whereas someone else may like classical music when resting.
- A users listening history can be used to identify contextual preferences for a user, which is their taste in a particular context [17].

These observations are useful, since they solidify the reasoning that a recommendation system can be built which takes into account contextual preferences, and therefore can be made unique for a certain user at a certain point in time.

2.2.2.3 Evaluation

One thing this paper states is *"With the popularity of mobile devices like smartphones, users can listen to music anytime and anywhere, which makes it difficult to acquire the real-time contexts directly"*. However, in modern times this is less true. Smartphones, along with other devices such as smartwatches, allow many data to be collected, which can be combined to accurately estimate a context in real time. This is show in the National University of Singapore paper [14], where even with very limited data (time, accelerometer data and microphone data), an activity could be accurately predicted. The Singapore paper was written in 2012, and this paper by Wang et al. [6] was written in 2018, after devices such as smartwatches had become mainstream. This means there is much more data available which, if following the learnings of the Singapore paper, means activities could be learned even more accurately. This would therefore give a method for classifying some contexts, specifically activities in this case.

Some useful observations from the paper are that music from the same artist tends to have low-dimensional representations which cluster together tightly, and the same with artists and songs in similar genres. This means that if a user likes a particular artist in a particular context, they're likely to like other songs by this artist. This therefore could lead to the idea of creating single-artist playlists for a particular context, e.g. 'Ariana Grande songs to run to' or 'Adele songs to study to'.

Another useful observation was that songs listened to by a user form one or multiple clusters, and songs in a user session cluster tightly. This reinforces the idea of using low-dimensional features to groups songs, since using actual user data, this paper showed that there was a mathematical pattern between songs which users listened to in a similar context, and therefore this can be used to recommend songs.

The final useful observation was that the more dimensions that are added to a song, the more stable the performance of the system became. This is because more features of the song could be captured, allowing the song to be more accurately represented. This worked up to a point of about 200 dimensions, at which the system would become less efficient and not increase accuracy.

2.2.3 MusicalHeart: A Hearty Way of Listening to Music

2.2.3.1 System Overview

Musicalheart [18] is a system designed in 2012 which uses a heart rate sensor in a pair of headphones to monitor heart rate and activity level, and use this to recommend songs in order to keep the heart rate at an optimal level. It does this by sending data to a server and dynamically suggesting songs based on the data it is receiving. It consists of three main parts: the *Septimu* platform, which is the heart rate sensor in the headphones, the Android application *MusicalHeart* for heart rate algorithms and server communication, and the web server, which handles the music recommendation.

The system works by first gathering heart rate and activity level data from the *Septimu* sensor. The design and testing of this sensor was a major part of the project. However with the increase in advanced devices which gather this data now widely available, it doesn't make sense to create a whole new sensor to gather this data.

The data is then processed on an Android phone in *MusicalHeart* app. The app received data streams from the sensor, as well as GPS and WiFi scan results. These then calculate the heart rate, activity level and context of the user , and then this processed data is combined into a time series of 3-tuples. It holds the most recent 10 minutes of tuples in order to have an overview of the users current state. The music recommendation system on the server then recommends a song based on this state, using a target heart rate which can be set by the user. It uses a biofeedback loop in order to measure the heart rate and use music to adjust it.

All personal data is calculated on the user's device, however it is sent to a web server in order to generate the music recommendations. This therefore means user reactions to songs can be shared to make the recommendation system more accurate. How this data is stored and kept secure and private is not described, however is something which should be considered in this type of system.

This could be done by using methods such as differential privacy, in order to make sure users cannot be traced from their data, or a strong enough encryption/on-device storage of sensitive data to make sure it cannot be leaked.

2.2.3.2 Observations

The paper did an empirical study with 37 participants in order to test how accurate the application was. It measure heart rate at all three levels of activity both with and without participants listening to music. They differ the participation on occupation, age, sex, fitness and ethnicity. However, one critical aspect which they don't mention is music taste, which presumably differed because of these other aspects, however is not measured.

One observation the study found was that tempo had a greater effect in increasing heart rate when the listener is exercising at a high intensity with the rhythm of the music, so listening to synchronous music. However, pitch and energy had a greater effect on heart rather than tempo at a lower activity levels. This is why they used a linear combination of these three features, which correlates more with the intensity of the workout.

Another observation was that the heart rate of users was less accurate for more intense workouts. This is due to the sensory using audio signals, which were poorer in high intensity workouts since the contact of the earphone was looser. It resulted in an average error of about 7.5 BPM. This is still the case with smartwatches, as demonstrated in a study on the validity and reliability of Apple Watch heart rate monitoring completed in 2018 [19], however the error was much lower, at around 1.3 BPM. The MusicalHeart paper also states that IR sensors (which most smartwatches are) have a smaller error than their sensor.

When evaluating their activities level detection, they tested both single activities as well as sequences of activities, to check their system could detect when a user's activity level changes between the three levels. Their results were very accurate, with 99.1% accuracy in single activities and 96.8% accuracy in sequences. This therefore shows it is possible to classify to a small range of activity levels very precisely using their model.

The paper also stated that as more users used MusicalHeart, the quality of recommendations would increase. This is because it collects the history of people's responses to different songs, and so the more responses it can collect, the more accurate the data will be, and the more data it will have for a wider variety of songs, and therefore the better songs the algorithm can recommend.

2.2.3.3 Evaluation

One important feature of MusicalHeart is that it can only classify an activity into one of three intensity categories: low (lying or sitting idle), medium (walking) and high (jogging). It also uses some other calculated features, such as whether the user is indoors or outdoors, velocity they're travelling at, and accelerometers to determine if they're upright or lying down, however these still only give a very coarse-grained overview of the user's context. This, combined with the fact that the application only takes into account a very small subset of a user's listening history, and does not take feedback on song recommendations, means that the music suggestions do not have as much data to learn on compares to other systems such as the Sequence-based Music Recommendation System analysed earlier [6].

Another limitation of this application is that songs are only represented by three features - tempo, pitch and rms-energy. This is a very different approach to the Sequence-based Music Recommendation System analysed earlier [6], which used up to 200 different features, and showed that using more dimensions of a song gave a more accurate recommendation. This is an area which could have been expanded upon, since the system for comparing the similarity between two songs was very simple in comparison to the sequence-based approach.

The MusicalHeart system assumes users have listened to between 20 and 55 songs, however the average number of songs a person has listened to while using streaming services is much larger, since the average Spotify user listens to over 25 hours of content a month [20]. Streaming services

have changed the way people listen to music, and also allow much more data to be collected on they music individuals listen to. This is because, compared to services like radio, people are in control of what they can listen to, and therefore allow a musical profile to be built up based on their listening history, which can be used to learn much more accurately what they like to listen to in a given context. Therefore, in the decade since this paper was written, the ability to build a musical profile of a user has become a lot easier, and therefore this could be used to give more accurate recommendations.

Finally, this paper makes three statements about why smartwatches are an infeasible way of monitoring heart rate. However, in the 9 years since this paper was published, these are now obsolete. Firstly, it states "*the user has to carry an extra device while exercising*". While this is technically true, studies have shown that 32.1% of people wear a smart watch or health tracker in the UK as of 2019 [21], and this number is only set to increase over the next few years [22]. Secondly, it states that "*some of these devices require the user to wear a chest strap*", which doesn't apply to modern smart watches or fitness trackers. Finally, it states "*the best of these devices cost about $400, which is prohibitively expensive for the average person*". However, as we have seen by the percentage of the UK population with a smart watch, they're becoming more widely available, and are drastically reducing in price, with fitness trackers available costing less than £50. Therefore, using a smart watch is much more of a feasible solution in the current technology climate.

2.2.4 Spotify's Soundtrack Your Workout

In July 2020, Spotify released a webapp which allows users to create a custom workout playlist based off of their own music taste [23]. A playlist can be built for a specific type of workout by selecting different options, which generate a customised playlist of songs both known and unknown to the user.

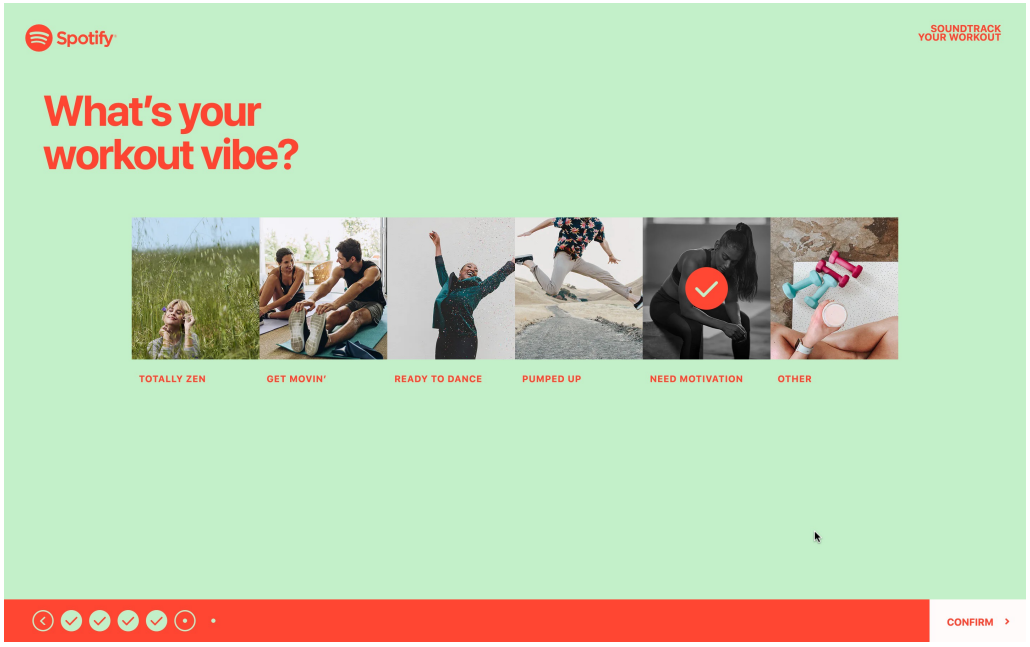


Figure 2.1: Pick the 'vibe' of your playlist

The app has 9 different customisable options - it first starts by asking how long the workout session should be, between 15 minutes and 2 hours. It then allows the option to have music, podcasts or both, and asks whether explicit content should be added. The next stage is to pick the workout type, which is limited to 8 basic workout types. The next stage is to choose who you're working out with, and then to pick the workout 'vibe' (as seen in Figure 2.1). Finally, you can pick either 1 or 2 genres the music should be selected from. You then have the option to upload a picture and give the playlist a name, and can review the selected options (Figure 2.2)and it's

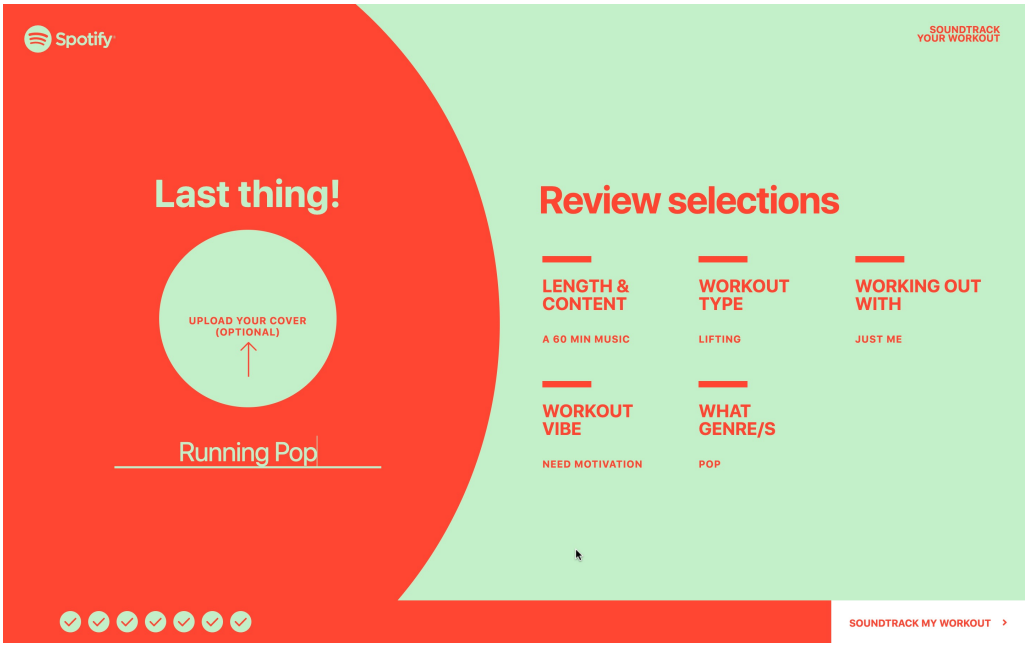


Figure 2.2: Generated Playlist Review Page

generated and automatically saved to your Spotify library (Figure 2.3).

One great advantage of this software is that it leverages the user's Spotify listening history in order to generate every playlist. This means that the majority of the songs will be familiar to the users, as they will correlate with their music taste. This is a benefit, since it's known that people perform better to exercise when they listen to music they are familiar with [2].

As seen by the images, the application also uses a very simple user interface, which is easy to navigate and uses very intuitive options at each stage. This makes it very simple for a user to generate a playlist, however also means it can be quite restrictive in the playlists which can be created. For example, if a user likes to listen to a particular artist when they're running, there is no option to generate a playlist fitting this criteria. This is where an application without these restrictions could find this pattern ans user it to generate more user-specific playlists.

One downside with this application is that that it requires a lot of configuration. One of the main purposes of automating playlist curation is that it takes a lot of time to curate the songs in the first place, but in this case, there are still up to 9 stages which need to be personalised by the user, and this needs to be done for every playlist they want to generate. This therefore requires these steps to be taken before each workout, rather than having the playlist generated automatically and waiting for when the user wants to start their workout.

Another downside is that the generated playlists are specific for exercising. This is a very common activity that people use music to motivate them for, however, as stated earlier, there are many more contexts where music can be helpful, such as sleeping. Therefore, since the application is only catering to a subset of potential activities, it is losing users who use music in different contexts.

The biggest drawback for workout-based recommendation is that Spotify has no access to a user's fitness data. This means that the playlist may be personalised for songs which they like, but these songs are not tailored to the users fitness history. For example, if a user always slowed down around 20 minutes into a run, then some more upbeat music may be needed to keep them on track, however this application has no way of knowing this.

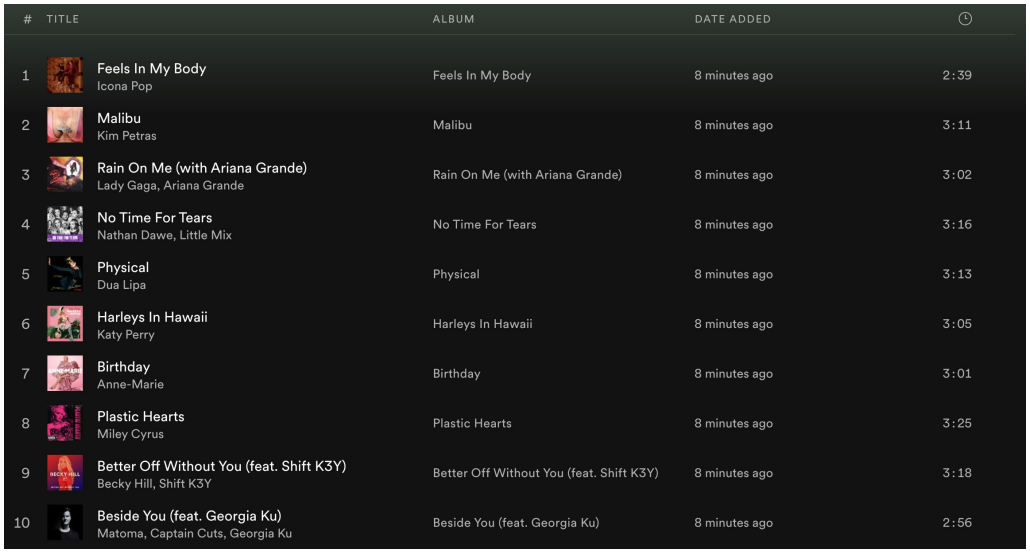


Figure 2.3: Generated songs

2.3 Background Summary

2.3.1 Common Themes

One common theme throughout the existing systems is that a finite set of contexts are classified. The sequence-based approach [6] doesn't put a limit on the number of contexts which a user can be assigned, however the system does not attempt to classify these by figuring out what they are. This therefore implies it is a challenge to figure out more than a handful of contexts.

Another common theme found was that adding more users to a system results in better recommendations. This can be seen, for example, in the National University of Singapore paper [14] This is because they use a version of collaborative filtering, which is where the listening patterns of one user can influence the recommendations of other users. Therefore, the more users in a system, the more data can be collected about in which contexts songs can be best played, and therefore means recommendations can be more accurate since they've got more data to learn from.

Also, the more data a system has on a user, the better recommendations it can give. This could either be by a user using the system for a long time, or being given a large amount of their listening history. This is shown in both the National University of Singapore study, as well as in the MusicalHeart paper, which both conclude that the longer the system is used, the more accurate their results were in their evaluation.

Another key observation was that more features that are collected about a song, the more accurately they can be represented. The Sequence-based Music Recommendation System used up to 200 different features. However, that study concluded that this was the limit to which increasing the number of features increased the ability for the system to accurately recommend songs. However, the MusicalHeart application only uses 3 features about each song: tempo, pitch and rms-energy, suggesting that if the features are well picked, they can still gain good results.

2.3.2 Common Problems

One common problem is that the majority of these solutions try to generate song recommendations in real time, and through doing this, limit the amount of processing power which can be used, as well as only being able to recommend one set of songs. The exception is Spotify's Soundtrack Your Workout, which generates a playlist. The real-time solution is useful since it they don't require user input, and are much more automatic than Spotify's solution. However, this also means that the recommendations are not stored, and so if a user has decided prior to playing music what type of activity they want to do, the system still needs to figure that out. However, the spotify solution

requires lots of user input, and therefore adds a delay to starting a workout, since the user needs to select information about their playlist. This is potentially duplicated from when they enter this information into a fitness tracking app, as well as some of it relating to music tast, which could be inferred from previous behaviour.

Another common problem is that the majority of solutions do not have access to a large data set of user listening history, or a large data set of workout history. This therefore means that when the program initially starts, the recommendations are not very accurate, since there is not a lot of data to learn from. The only exception to this is the Spotify application, which has access to a user's full listening history. However, since this application does not integrate with any kind of fitness tracking, it's limited in how personalised the playlists can be, since their only source of information on the user is music taste.

An important theme throughout the literature is the limited ability to measure success of the systems. Since every user has a unique music taste, it's difficult to have a single measurement to compare all results against. Therefore, some of these systems, such as The National University of Singapore's system [14], compare against randomly recommending songs. Comparing different internal algorithms, as well as other systems, could therefore produce a more substantial evaluation.

Chapter 3

Implementation

The aim of this project is to generate music recommendations to users for use during workouts, primarily in the form of pre-generated playlists, and also in real time. This culminated in the creation of an application named Osti.

3.1 Overview

Figure 3.1 shows the different components which make up the Osti system, including the servers which fetch listening and workout data from the relevant APIs, compute recommendations and host the web interface. It also uses a MongoDB database hosted using the MongoDB Atlas cloud database service. A schema for this database can be seen in Appendix A.

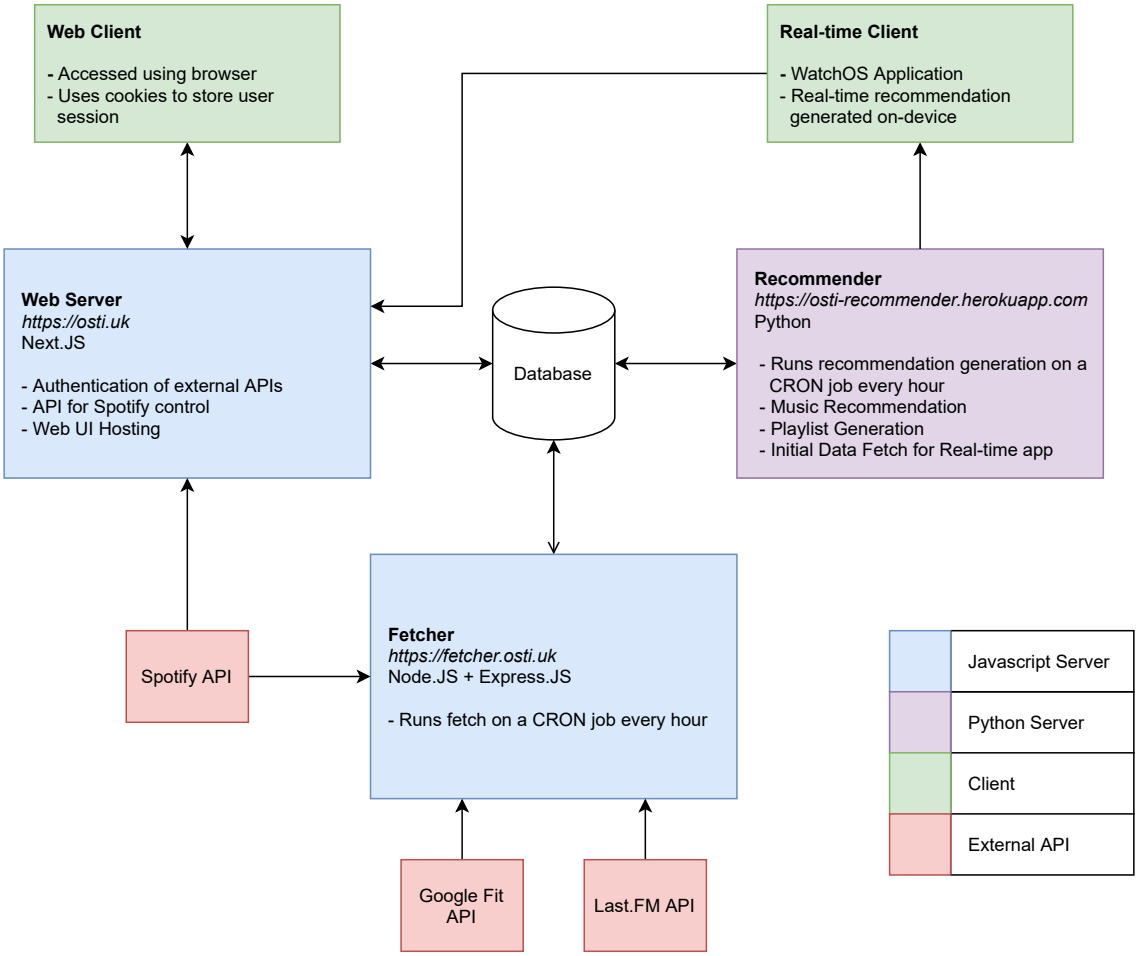


Figure 3.1: System Architecture Diagram

3.2 Data Fetching Pipeline

The first stage of the application is data collection. In order to be able to reach the largest number of people, the service we decided to use to collect information about music listening history is Last.FM, since it can connect with multiple streaming services. These include Youtube Music, Spotify and Apple Music, which together make up 60% of the music streaming industry [24]. This allows the maximum potential number of users to use the application, and gather the most data for the system to learn from. An alternative choice was to user the Spotify API, however this only gives access to the 50 most recently played songs, whereas Last.FM has an unlimited history.

To collect workout data, the Google Fit API was chosen, since this can connect to a large majority of smart watches, including Apple Watch, which dominates the market with over 55% market share [25], as well as most other devices such as FitBit.

The data fetching pipeline can be split into two main events; the initial data fetch, where as much historical data as possible from the APIs are collected, and the regularly scheduled updates, which run once every hour to make sure all user workouts and listening history in the system is kept up to date.

The fetching service is hosted on a separate machine hosted using Heroku, for two main reasons: to reduce load, since having it with the web server and recommendation generation all on one server could cause it to be overloaded, as well to remove a single point of failure. This means that if this service were to go down, existing data would still be able to be used to generate recommendations for existing users. It uses an Express.JS server, hosted at `https://fetcher.osti.uk`.

This microservice was written using Node.js, since the APIs used both return JSON files. It also lead to the decision of using a MongoDB database, which also uses a JSON-like structure to store documents, and is flexible with different sized data, for example workouts of differing length.

3.2.1 Connecting to APIs

The first step of data collection is for the user to link their Google and Last.FM accounts to the application, in order for data to be regularly collected. This involves storing an access and refresh key, which is sent with API requests to prove the application is authorised to access the data. These are stored in the 'Users' database collection.

To keep track of which stage the fetcher is at, an enum is stored in the database, attached to the user's profile. These states are shown in a state transition diagram in Figure 3.2. This prevents race conditions, for example two fetches happening simultaneously, since the value is checked and updated before any action takes place.

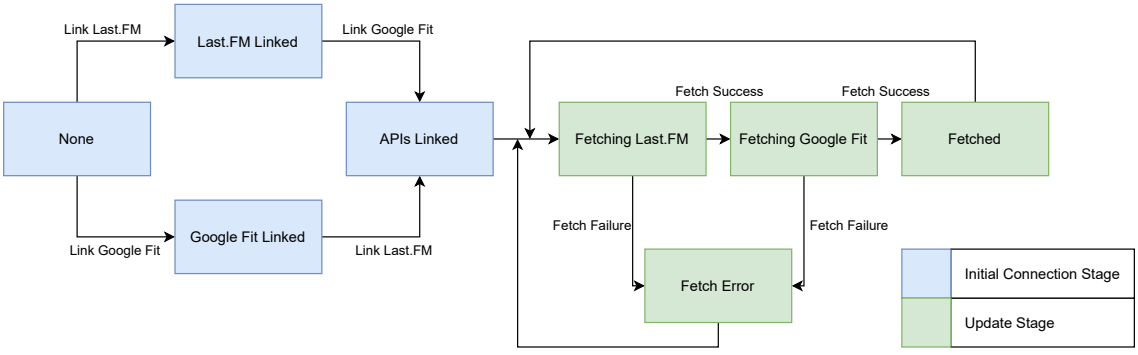


Figure 3.2: State Transition Diagram of fetcher states

3.2.2 Listening History Collection

Once access has been granted, the process of data collection begins. On the first run of the fetching, a users entire LastFM history is fetched, which consists of some generic track information, as well as when it was listened to. This gives the maximum amount of context possible for the system to infer the user's music taste from, even if some of these songs have not been used in a workout, since it's been shown that users workout better to songs they're familiar with [2].

These listens, referred to by Last.FM as 'scrobbles', then have their Last.FM identifier used to check if the track already exist in Osti's database. For any unknown track, the Spotify Search API [26] is used to collect basic information such as artists, album, duration and Spotify identifiers. If the song cannot be found on Spotify, a track is still created in the database, just without any of the information which would be retrieved from Spotify, and the user has the ability to connect this manually from the web interface, described in Section 3.6.5. This is done by simply copying the share link for the song from Spotify, however, is rarely needed, since at the time of writing, only 1186 of 14780, around 8%, of tracks found through this method were not able to be identified, and in many cases this was because these songs are not available on Spotify, and have been scrobbled from other sources.

Once the tracks have been located on Spotify, the API is used again to get the audio features for each track. These features, and there descriptors, are described in Table 3.1. In the rare case that Spotify hasn't generated features for a song, which at time of writing has occurred for 8 of the 180998, or 0.004% of tracks, then the fields are left blank, and the songs are not recommended until the features can be identified.

These tracks then have their features saved to the database, along with the Last.FM scrobbles, with a reference to either the pre-existing or newly created track with features in the database.

This process then repeats hourly in order to collect scrobbles as soon as a user listens to music, however the API is only queried back to the last recorded scrobble in the Osti database. This decision was made, instead of recording the last request, in case the Last.FM API doesn't always update instantly.

3.2.3 Workout History Collection

The next step is to fetch workout data from the Google Fit API. This data is limited to the previous year by Google, however all the available data is collected, even for workouts where music wasn't listened to, since it allows the system to later build a clearer image of user workout features, such as average workout length, heart rate, or calories burned, during a particular workout type.

Once these workouts are fetched from the API, another call is made for each workout to collect more detailed data points at 10 second intervals. This time value was chosen since shorter intervals wouldn't increase accuracy, and instantaneous values such as heart rate and speed, don't fluctuate massively every 10 seconds, so this level of detail gives a good overview of a workout. Other values, such as calories and distance travelled are cumulative, and therefore the interval time doesn't make a difference.

The workout object initially retrieved is then augmented with this data, and added to the database.

3.3 Music Recommendation Engine

The second major stage of the implementation was to create a system which suggested the ideal tracks for each user for a specific workout. These tracks would then go on to either be arranged into a playlist, or provided to an application which would recommend the best of these songs in real time.

Recommendations are able to be generated as soon as a user has listened to a single song during a particular workout. They become more accurate over time as more songs are added, but the use

Feature	Description	Data Type
Acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic, where 1.0 is high confidence.	Float
Danceability	How suitable a track is for dancing, based on a elements such as tempo, rhythm stability, beat strength, and overall regularity. Values range from 0 to 1, where 1 is most danceable.	Float
Duration	The length of the track in milliseconds.	Integer
Energy	A perceptual measure of intensity and activity, where energetic tracks feel fast, loud, and noisy, based on features such as dynamic range, perceived loudness, timbre, onset rate, and general entropy.	Float
Instrumentalness	Whether a track contains vocals, so the closer the value is to 1.0, the more likely the track has no vocals.	Float
Key	The key the track is in, where integers map to pitches using standard Pitch Class notation, such that 0 = C, 1 = C♯/D♭, 2 = D, and so on.	Integer
Liveness	Dtects the presence of an audience in the recording, where a value above 0.8 provides strong likelihood that the track is live.	Float
Loudness	How loud a track is, averaged across the track, in decibels (dB), usually ranging between -60 and 0 dB.	Float
Mode	Whether a track is major or minor, where major is represented by 1 and minor by 0.	Integer
Speechiness	The presence of spoken words in a track, where values above 0.66 describe spoken word tracks, and values below 0.33 represent music and other non-speech-like tracks.	Float
Tempo	The overall estimated tempo of a track in beats per minute (BPM).	Float
Time Signature	An estimated overall time signature of a track, which is how many beats are in each bar.	Integer
Valence	A measure describing the musical positiveness conveyed by a track, where values closer to 1.0 sound more positive.	Float

Table 3.1: Description of the audio features used to represent tracks [27]

of track features listed in Table 3.1, along with collaborative filtering from version 2 of the engine onwards, allow a large number of recommendations can be made from even a small subset of user listening history for that workout.

All 5 versions are generated regularly once a day, however only the final user-feedback version is shown to the user, since it builds upon the majority of the others. This can be updated in frequency, however through user testing it was discovered the rankings don't change very much on a smaller scale. The only exception to this is with the introduction of user feedback, where songs can be given a positive or negative boost. In this case, the engine can be triggered more frequently.

The system is hosted on a separate Heroku machine, and is written in Python using a Flask server, along with Celery for task management. The separate server, as before, means that it doesn't interfere with the fetching of data from the API, or displaying information to the user via the web interface. The use of Celery for task management also means that tasks can be queued if multiple requests are sent at once, which is useful for when users can trigger recommendations, in order to prevent machine overload.

We'll now look at the iterations of this algorithm, and the techniques used to provide the most appropriate recommendations for each user-workout pair.

3.3.1 Version 1: Popularity Based

The first version of the recommendation engine was the simplest implementation. It is a count-based method which returns the most frequently listened to tracks for a user during a particular workout as the recommendations. This is a good baseline to compare other iterations of the engine against, and also was useful to display these results to the user so they can give boosts to some of their most frequently listened to tracks.

One advantage of this iteration is that it can run independently for each user, since the recommendations only rely on the history of the user who they're being generated for. However, for the explanation, we'll describe how we generate it for all users at once, since this will assist in the explanations of the further versions of the engine.

It works by downloading the workout and listening history for the target user from the database. The workout types, users and songs are then one-hot encoded, which is useful when creating the utility matrix of preferences. The utility matrix generated here is an extension of the idea described in Section 2.1.1, since another dimension is added, so it is indexed firstly by user, then workout type, then track. This is a sparsely populated matrix, but allows an easy access to which users are listening to which songs for each workout, and is used frequently in the literature [7].

Let the listening history of user i be l^i, which consists of n listens $l^i_1, ..., l^i_n$. The workout history is similarly defined as w^i, consisting of workouts $w^i_1, ..., w^i_n$. The start time of a listen is defined as $s(l^i_m)$, and the start and end times of a workout are similarly defined as $s(w^i_m)$ and $e(w^i_m)$. To simplify the model, the end time of listens are not considered at this stage.

We then iterate over the ordered l^i and w^i simultaneously, and if a listen falls within the time frame of a particular workout, so $s(w^i_m) < s(l^i_n) < e(w^i_m)$, then the appropriate value in the utility matrix is incremented. By iterating over these together, we reduce this problem to linear complexity, based on the number of listens they have. Once the utility matrix of preferences is calculated, then a maximum of the top 100 songs for each user and workout are selected and saved to the database as recommendations.

These recommendations were utilised later on in the user testing stage, since showing a user their most popular tracks for a particular workout will allow them to rank whether on not they agree they're something they want to listen to in that particular context

3.3.2 Version 2: Content-Based Recommendations

The second iteration of the music recommendation engine incorporates the features of a track described in Table 3.1. These features can be split into two sections: learned features, which have been predicted, for example, by a machine learning model, and definite features, such as tempo and key, which are known.

The utility matrix of preferences is calculated in the same was as in version 1 of the algorithm, however the indexing order is switched, so that it's indexed by workout type, then user, then track. This is useful, since versions of the engine onwards from here will use collaborative filtering (CF), and since CF is done on a per-workout basis, it made sense to group the user-track arrays by workout in order to optimise array accesses.

The next step is to generate the trackset T^u_w, which is the collection of tracks which could potentially be recommended to the user u for a particular workout type w. This is generated from the union of three smaller sets:

- t^u_w - A set consisting of the tracks listened to by user u during workouts of type w
- t_w - A set consisting of the tracks listened to during workout type w by all users
- t^u - A set of the tracks listened to by user u, both during not during workouts

t^u_w and t_w are generated using the utility matrix of preferences, and is generated directly from the database data, since it doesn't rely on workout data. They each also contain a popularity

metric, which represents the frequency at which they're listened to in their respective set. This is used later so that more commonly listened to songs have a higher probability of being recommended.

These three sets are then used to form what the overall trackset for user u and workout w, T_w^u, where:

$$T_w^u = t_w^u \cup t_w \cup t^u$$

To find the ideal recommendations from this trackset, the first step is to find a representation, r_w^u, of the music the user u usually listens to during the particular workout type w. This is calculated using t_w^u, by taking a weighted average of each of the features listen in Table 3.1 for each track, weighted by the frequency they've listened to the track during that workout. This means that a song which is listened to frequently by the user during that workout will contribute more to the representation that one which has only been listened to once.

The final stage is to find the ideal songs from trackset T_w^u for the user-workout combination. This is achieved by calculating the cosine similarity between the representation r_w^u and every potential track in T_w^u (as described in Section 2.1.5.2), and recommending the tracks which have the greatest cosine similarity. This can be formalised as:

$$\text{csim}(r_w^u, (T_w^u)_j) = \frac{\sum_{i=1}^{n} (r_w^u)_i \times ((T_w^u)_j)_i}{\sqrt{\sum_{i=1}^{n} ((r_w^u)_i)^2} \times \sqrt{\sum_{i=1}^{n} (((T_w^u)_j)_i)^2}}$$

To calculate the k values with the greatest cosine similarity, we can repeat the following equation, in each iteration removing the value with the greatest cosine similarity from T_w^u:

$$\underset{j}{\text{argmax}}(\text{csim}(r_w^u, (T_w^u)_j))$$

The resultant k tracks with the greatest cosine similarity to r_w^u are then returned as the recommendations for user u and workout type w.

3.3.3 Version 3: Hybrid Collaborative Filtering and Content-Based Recommendations

The previous version of the music recommendation engine had an unintentional element of collaborative filtering, since songs which other users had listened to during the same workout type recommendations were being generated for were considered in the trackset the recommendations are being source from. However, with a small user base, there was such a small overlap in songs listened to by multiple users that this wasn't really being utilised in the recommendations (as of time of writing, only 113 tracks had been listened to by more than 2 users).

Therefore, the solution was to introduce an external data source that count introduce many new tracks into the system which can be recommended to users. This allows new tracks to be discovered by users, and since there are many more tracks in the system, these are more likely to have a higher similarity to their ideal workout song, and therefore will be more suited than the limited number of tracks which exist in the system.

Two main potential datasets were considered to incorporate this data into the system. However, finding the most appropriate dataset was challenging, since there is no explicit dataset which links listening history to context, since the are has not been researched very extensively. The first dataset considered was the Spotify Music Streaming Sessions Dataset [28]. This consists of around 130 million listening sessions and their corresponding user interactions. However, there way no way to link these sessions to a particular workout, and therefore it wouldn't be very useful in the context of collaborative filtering with respect to workout type.

Therefore, the dataset chosen was the Spotify Million Playlist Dataset [29, 30]. As is implied by the name, this dataset contains a million user-created Spotify playlists, including their name and

Workout Type	Number of Playlists
Running	7408
Sport	4346
Sleep	2577
Yoga	792
Dancing	567
Walking	445
Cycling	131
Swimming	76
Strength Training	72
Pilates	28
Rowing	11
Mindfulness	10

Table 3.2: Frequency of Workout Types found in Million Playlist Dataset

full, ordered tracklist. This meant that, despite the playlists not explicitly representing a context, since many users create playlists for working out, and name them with something similar to the workout type the playlist is created for, these could be inferred as songs which at least one user finds appropriate to listen to for the corresponding workout. Out of the million playlists, 16,997 contained a common workout type in the name, with the cumulative length of these playlists being 913,726 tracks, of which 168,222 were unique. The distribution of these tracks for each workout can be seen in Table 3.2.

These playlists were incorporated into the version 2 of the music recommendation engine to increase the number of tracks which could be recommended for each workout type. This was done by treating each playlist as a workout by a different user, since there was no way to track if multiple playlists belonged to the same user, due to the anonymisation of the dataset. These then meant that the subset of the trackset t_w, which consists of the tracks listened to during workout type w by all users now consists of many more songs, and these are all potential recommendations to the user.

3.3.4 Version 4: Reintroducing Popularity Metrics and Categorical Features to the Hybrid System

The next stage was to introduce the popularity metric mentioned in the creation of version 2 of the engine. This metric is calculated slightly differently for the three subsets of the trackset t_w^u, t_w and t^u.

For t_w^u, the ranking of track $(t_w^u)_i$ is:

$$2 + (1.5 \times \frac{\text{count}((t_w^u)_i)}{\max\limits_j (t_w^u)_j})$$

Where $count((t_w^u)_i)$ is the number of times user u has listened to track i during workout w.

For t_w, the ranking of track $(t_w)_i$ is:

$$\frac{\text{count}((t_w)_i)}{\max\limits_j (t_w)_j}$$

Similarly, for t^u, the ranking of track $(t^u)_i$ is:

$$\frac{\text{count}((t^u)_i)}{\max\limits_j (t^u)_j}$$

Where count is defined similarly to t_w^u in both cases. The ranking is increased by constant factors for t_w^u since it's been shown that songs which a user knows are more effective in motivating

Boost Value	Effect
-2	Track should never be recommended for this workout type
-1	Track should be recommended less frequently for this workout type
0	No boost
1	Track should be recommended more frequently for this workout type
2	Track should always be recommended for this workout type

Table 3.3: User feedback loop boost effects

them during exercise [2], and also that users have a specific music taste for different workouts [15, 16], and therefore the tracks in t_w^u are better recommendations for the user. These metrics are then used as another weighting for the predictions, and result in tracks which are listened to the most frequently being more likely to be recommended, since they're known to be liked in this context.

In this version of the music recommendation engine, we also added in track artist as a categorical feature for each song. This is done by creating an artist map for each user-workout pair as recommendations are being generated, which contains how many tracks by each artist a user has listened to in the corresponding context. This is then used as an extra factor in weighing predictions, since it's been shown tracks which a user has more familiarity with are more likely to increase workout performance [2]. Add one smoothing was used here to make sure that artists which are not listened to frequently still appear in the recommendations, however are just not weighted as highly.

Adding in album here was also considered, however it was decided against due to the increase in computational power required, along with the fact that albums are usually performed by one artist, and therefore would just be a repeat of the calculations for artist weightings discussed above.

The method of calculating the artist multiplier is similar to the Jaccard Index described in Section 2.1.5.1. However, instead of representing each artist as a binary of whether the user listens to them, the number of listens to that particular artist is taken. Then to calculate a track's artist multiplier, the sum of all artist totals in the track is taken, and then divided by the maximum artist total. This was chosen instead of the sum of all listens since it gives the multiplier a value closer to 1, therefore not reducing the score as much. This therefore results in a maximum multiplier as the number of artists on the track, however realistically this will be a number smaller than 1, since most tracks have one or potentially two artists.

3.3.5 Version 5: Adding a User Feedback Loop

The final version of the music recommendation engine incorporates a user feedback loop into the algorithm. The reasoning behind this was that recommendations are subjective, and despite usually attaining a good prediction from the methods above, users may wish to alter these suggestions. There may be a predicted track which a user doesn't think is appropriate for a playlist, or on the other hand, one of their favourite tracks for a particular context may not be included.

Therefore, the solution was to add a user feedback system, which is early on in the recommendation pipeline, and would allow users to alter any tracks for which they do not agree with the recommendation score. The loop was introduced in this stage, rather than at a the later playlist generation or live recommendation stages, since it means that users don't need to worry about altering track suggestions during their workout, and also adds less volatility to the recommendations since the set number of recommendations given to these later services will still be maximised.

A piece of user feedback is referred to as a boost b, and can be given one of 5 values, described in Table 3.3.

These boost values are taken into account in the final stage of the music recommendation engine, appropriately moving a track up or down in the order of recommendation such that tracks appear to the user's preference. The formula used to move these tracks is as follows:

$$\text{boost}((T_w^u)_i) = \max(0, s((T_w^u)_i)) + \left(b((T_w^u)_i) \times \frac{\max_i(s((T_w^u)_i))}{2} \right)$$

Where $s((T_w^u)_i)$ is the score generated by version 4 of the algorithm for user u, workout w and track i, and $b((T_w^u)_i)$ is the boost value assigned to the same song.

This equations essentially moves any tracks with a +2 boost to the top of the recommendations, and removes any tracks with a -2 boost from the set of recommended tracks. Songs with no boost are not effected, and songs with a +/- 1 boost are move up or down the recommendation list by half the score of the top recommendation, which usually results in a significant move.

The interface for users adding these boosts is discussed in Section 3.6.2.

3.4 Playlist Generation

The next stage was to generate the 'ideal' playlist for a particular workout. This uses the top songs recommended by version 5 of the music recommendation engine in order to bring the workout vitals of heart rate, calories burned, distance travelled, speed and number of steps closest to the user's target values. By default, it uses target values for these workout features that are predicted by the system, however these can also be manipulated by users. From these, it finds the most appropriate songs to keep a user on target for the workout, based on metrics such as their historical effect on the workout features, tempo and positioning in in the list of recommended songs.

The Heroku machine used for the music recommendation engine is shared for this service, since they're usually executed sequentially, and are both written in Python.

3.4.1 User Workout Targets

The first stage of generating a playlist for a particular user-workout pair is to calculate the target values of the following 6 workout features:

- Workout Length
- Number of Calories Burned
- Heart Rate
- Distance Travelled*
- Speed*
- Number of Steps Taken**

* - This feature is only used for workouts which require some kind of movement, such as indoor or outdoor running, cycling or rowing, where data is available.

** - This feature is only used for workouts where some kind of on-foot movement occurs, such as walking or running.

The initial values of these targets are calculated by simply taking an average of each feature for each workout, and then calculating the average of these collective averages. These values (excluding workout length and heart rate) are then multiplied by a multiplier, which has a default value of 1.05, in order to continually push the user to increase the effort put into their workout.

However, these averages may not be useful for every workout - for example, if a user likes doing both 5km and 10km runs, then their average may be somewhere in the middle. Therefore, all of these values are able to be customised by the user. This can be done on the web interface, as described in Section 3.6.3. Multiple features can be adjusted to fit a user's desire, and the playlist generation algorithm will adapt to generate playlists to more accurately keep a user to these targets.

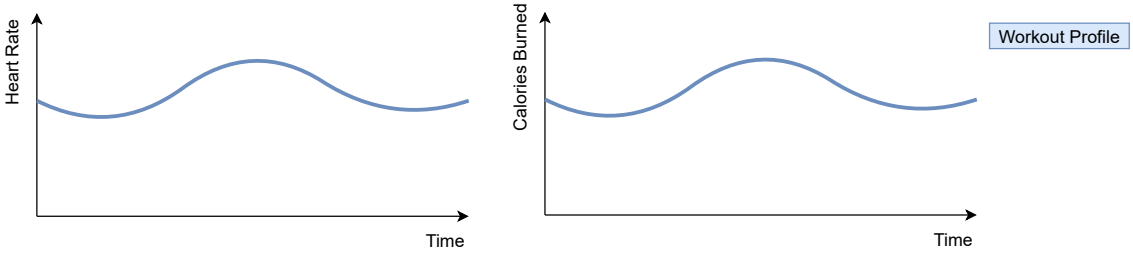


Figure 3.3: Workout Profiles showing the dimensions of Heart Rate and Calories Burned. Left: Instantaneous Feature (Heart Rate), Right: Cumulative Feature (Calories Burned)

These targets are generated whenever a playlist is created, and are updated in the database, so the most recent values are always available to the user. If a user has overridden these values, those are displayed and used, however if they reset them, the values don't need to be recalculated as a copy of the current base ones are kept in the database.

3.4.2 Workout Profiles

The next stage of playlist generation is to find patterns a user follows during a particular workout type. This will allow a profile to be built up of how a user behaves on an average workout with respect to up to five of the features of heart rate, calories burned, distance travelled, speed and steps taken. To find these patterns, each historical workout of a particular user is analysed, and for each 10 second interval, the average of these features is averaged over each historical workout.

Once these workout profiles are created, they need to be scaled down to be the same length as the target workout length. There were two methods considered to do this: scaling the entire profile down to this target length, or slicing the profile at the desired length. The upside of the first method was that it kept all of the data. However, since the data is being stretched, it won't accurately represent what actually happened in a workout. For example, if we have a 60 minute workout that we want to scale to a 30 minute target workout, then the total calories would stay the same, however it would be expected to burn them in half the time, which is not reasonable. Therefore, the option to slice the profile at the desired length was chosen.

With this decision, there is the case where the target length is longer than any previous workout. In this case, since there is no data for feature values at this time, the average feature values calculated in Section 3.4.1 are used instead.

The workout profile P_w^u for user u during workout w can be formalised as:

$$P_w^u = (p_w^u)_0 ... (p_w^u)_n$$

Where n is the target length of workout of type w for user u, and the value $(p_w^u)_i$ is a vector of up to 5 dimensions, one for each workout feature which is available for the workout type w, representing the average value of that feature from time point i to $i + 10$, measured in seconds.

Figure 3.3 shows a basic example of how a workout profile could be visualised over time for heart rate (an instantaneous feature) and calories burned (a cumulative feature).

3.4.3 Workout Deltas

Once a profile has been built for the user-workout pair, the difference between the workout profile and target value at each time point throughout the workout needs to be calculated. The first thing we need to do for this is to generate a target vector, where we spread the user-workout targets from Section 3.4.1 over the target workout length. This is done differently for instantaneous features (heart rate and speed) and cumulative features (calories burned, distance travelled and steps taken).

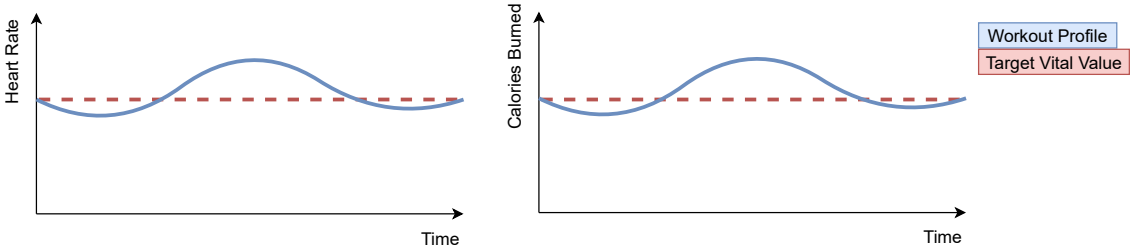


Figure 3.4: Target Heart Rate and Calories Burned over time. Left: Instantaneous Feature (Heart Rate), Right: Cumulative Feature (Calories Burned)

For instantaneous features, the target value will stay constant throughout the whole workout. For example, the if the user's target heart rate is 110bpm, then the target at each 10 second interval will also be 110bpm, therefore the target vector is trivial to generate.

For cumulative features, the target value is a sum over the whole workout. For example, if the user wants to burn n calories over a workout of m minutes, then they will need to burn $\frac{n}{6m}$ calories in each 10 second interval, and the vector of length $6m$ will have each value as $\frac{n}{6m}$.

The implementation of this section is designed to keep the system extensible, by using target vectors and not just a single value. A possible extension where targets do not need to be constant over time is discussed at the end of this paper in Section 6.2.1.

The target vector can be formalised as:

$$\mathcal{T}_w^u = (\tau_w^u)_0 ... (\tau_w^u)_n$$

In this case, all values $(\tau_w^u)_i$ will be identical, however as mentioned above this allows for the targets to be extended to also include different workout types where targets change over time.

Figure 3.4 shows the target vitals visualised on the same graph as in the previous section.

The next step is to work out the difference between the target of each workout feature, $(\tau_w^u)_i$, and the equivalent value in the workout profile at the same time point $(p_w^u)_i$, where both of these are vectors of the same size, since they hold the same features. This again is calculated slightly differently for instantaneous and cumulative features.

For instantaneous features, we calculate the 'delta' between the target vector at time point i, $(\tau_w^u)_i$, and the equivalent value in the workout profile $(p_w^u)_i$ simply by calculating $\delta_w^u = (\tau_w^u)_i - (p_w^u)_i$. This results in the change in each instantaneous value over the 10 second interval. Using heart rate as an example, at time point 0 the user's heart rate in the workout profile is 110bpm, and the target value is 125bpm, then the delta at time point 0 for heart rate is 125bpm - 110bpm = 15bpm.

For cumulative features, calculating this delta doesn't make as much sense. This is because we sum the deltas over time, and therefore they'll usually be the same for a particular workout over a particular time period with a similar amount of effort put in from the user. Taking calories burned as an example, over a 10 second interval, we want to find the number of calories which need to be burned, which can be calculated by $\delta_w^u = 2(\tau_w^u)_i - (p_w^u)_i$. For example, for the interval between time point 0 and 10, the target is to burn 2 calories, however in the workout profile, only 1.5 calories are burned. This means to compensate for this under-performance, we want to try and burn 2.5 calories in this interval. This will then average out over time such that the target value for number of calories burned is reached, and is illustrated in Figure 3.5.

The overall set of workout deltas for a workout of type w, for user u, and of target length n is formally represented as:

$$\Delta_w^u = (\delta_w^u)_0, ..., (\delta_w^u)_n$$

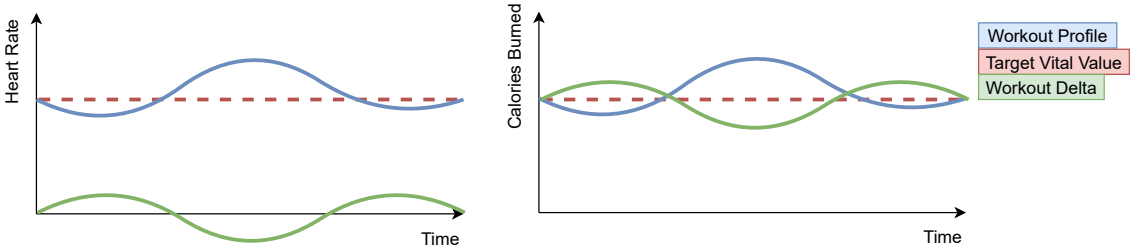


Figure 3.5: Workout Deltas for Heart Rate and Calories Burned over time. Left: Instantaneous Feature (Heart Rate), Right: Cumulative Feature (Calories Burned)

Figure 3.5 shows the workout deltas visualised on the same graph as the workout profiles and target values. It shows how the deltas of instantaneous features (heart rate, left) are the opposite of the difference between the target and workout profile, centered around zero. However, for cumulative features (calories burned, right), the workout deltas is centered around the target value. This is because these features are summed up over time, and therefore the number of calories burned over a subset of time is useful, as this can directly correlate to the number of calories burned over the duration of a song. However, for instantaneous features such as heart rate, the effect the song has on heart rate (increase or decrease) is more useful than the exact heart rate change.

3.4.4 Song Deltas – Coarse Grained

The penultimate stage is to calculate the equivalent deltas as above on the songs which have been listened to during workouts. These deltas are then stored in the database so they they can be used by other users who have that same song in their recommendation set, since the effects of a song on one user is usually similar for other users. This also means they only need to be calculated for any new workouts which enter the system, which will be since the user last generated a playlist for the current workout type.

These deltas are calculated in a different way to the workout deltas, and again are calculated differently for instantaneous and cumulative features. For instantaneous features, the value of the feature at the start of the song, s and at the end of a song, e are taken, and the delta is calculated by $e - s$. So for example, if a user's heart rate was at 110bpm at the start of a song, and 115bpm at the end, then the song delta for heart rate would be +5bpm.

For cumulative features, the sum of the feature over the length of the song is calculated. So for example, if a user burns 30 calories over the duration of a song, then this value would be assigned as the calorie delta.

The decision to make these deltas coarse grained was to reduce the amount of data which needed to be stored in the database, and allowed for faster playlist generation at the final stage, since only one comparison per song was needed rather than multiple. However, a more fine grained approached, where the songs deltas are computed at 10 second intervals, is explained below, and could potentially be used in further iterations once more data becomes available, or when more computing power is available.

3.4.5 Song Deltas – Fine Grained

Since we can also gather the effects of a song on workout features at 10 second intervals, we can take a more fine-grained approach in order for the songs to be more accurately predicted. In this case, the links between the different parts of a song and a user's workout performance can be more accurately detected, leading to more accurate predictions, to keep users at their ideal performance.

The differences to coarse grained song deltas are that the deltas are calculated at each 10 second interval, rather than across the whole song. For instantaneous features, the delta at time point i is calculated by measuring the change in a features value from time point i to time point

i+9 (since we use 10 second intervals). Taking heart rate as an example, if a user's heart rate is at 110bpm at time point 0, and 115bpm at time point 9, then the delta assigned to time point 0 would be +5 bpm, indicating that those first 10 seconds of the song increase the heart rate by 5bpm.

For cumulative features, the delta at time point i is calculated simply by finding the sum of the feature between time points i and i+9. So for example, the delta at time point 0 for calories burned would be the sum of calories burned from time point 0 to time point 9, indicating how many calories are usually burned within the first 10 seconds of the song.

3.4.6 Using workout deltas, song deltas and recommendations to generate a playlist

The final stage of playlist recommendation is the generation of the ordered list of tracks using the workout and song deltas described in Sections 3.4.3 and 3.4.4 respectively, and the recommendations from the music recommendation engine described in section 3.3.

Firstly, the recommendations generated earlier by the music recommendation engine for the current user-workout pair are retrieved from the database, as well as the coarse-grained song deltas for any song in this recommendation list. There are 4 types of song data which are used, which are described in Table 3.4.

	Same Workout	**Different Workout**
Same User	0.5	0.2
Different User	0.2	0.1

Table 3.4: Song Delta Combinations and Weightings

If a user has listened to the current song during the workout type the playlist is being generated for before, then those deltas are used, since they are most relevant to the user. However, since recommended songs have not always been listened to by the user, or may have been listened to for a different workout, this cannot be relied on. Therefore, track recommendations whose deltas have wither the workout or user in common are considered, however are weighted lower than the original deltas. Recommended tracks for which no deltas exist with either the same user or workout are weighted even lower.

Once these deltas have been collected, the next stage is to predict which song out of the potential 200 returned by the music recommendation engine should be queued next in the playlist. To do this, we keep a pointer of what position in the playlist we are at, which can be represented by p. Then, we iterate over the set each potential song, represented by S, and compare the workout deltas over the length of the song, $(\delta_w^u)_p, ..., (\delta_w^u)_{p+l(s_i)}$ to the song delta of that song $d(s_i)$, where $l(s_i)$ is the length of the ith song, and $d(s_i)$ is the song delta of the ith song.

To compare these deltas, we calculate the equivalent deltas from the workout deltas, such that for instantaneous features, we find the change in feature from the start of the song to the end of the song, and for cumulative features, we find the sum of the feature over the duration of the song. We then use the cosine distance as defined in Section 2.1.5.2 to find the distance between the workout delta and song delta for each song. This is repeated three times, for the set of deltas with the same user and workout, for the set of deltas which have either the same user or same workout, and for the set of deltas with neither the same user nor same workout. Any recommendation which does not have any deltas is given a distance of 1, which is the maximum potential distance.

The weighted cosine distances of all three possible sets are then added together in the ratio 5:2:1 as described above in Table 3.4, giving the highest weight to the deltas which were calculated by the user completing the appropriate workout.

However, this is not the only metric used to calculate which song should be queued next. The next metric incorporated is the difference between heart rate and song tempo. This is used since it has been shown that the tempo of a track can help to raise or lower a user's heart rate [31]. This is added as a penalty using the formula:

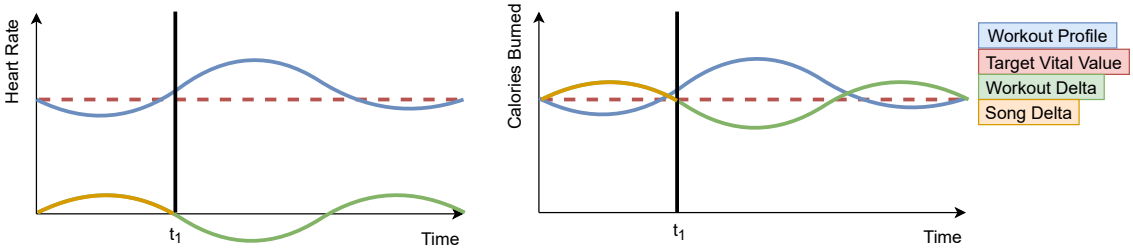


Figure 3.6: An example of a track being matched so a subset of the Workout Delta during Playlist Generation. Left: Instantaneous Feature (Heart Rate), Right: Cumulative Feature (Calories Burned)

$$0.3 \times |\frac{\text{Tempo} - \text{Target Heart Rate}}{\text{Target Heart Rate}}|$$

The other metric incorporated is the position of the song in the rankings from the music recommendation engine. This is added as a penalty to the overall score such that a song has a penalty of $\frac{n}{100000}$, where n is the position of the song in the ranking. The constant weightings of these metrics were empirically tested as to equally balance them.

Once all of these metrics are combined, the song in the recommendation set which has the minimum penalty is selected, and this is pushed to the playlist queue. The pointer p is then increased by the duration of the song, and the process is repeated until p is greater than the target length of the playlist.

Figure 3.6 shows a single track of length t_1, with song delta in orange, being matched to the workout delta in green. This track matches the workout delta closest, and therefore is queued next.

The playlist is then saved to the database, and can be viewed by the user on the appropriate recommendations page in the web interface, as well as be saved to their Spotify account, as detailed in Section 3.6.4. The 10 most recent unique playlists are saved for each user-workout pair. Playlists are evicted with a first in first out eviction policy. However, if a playlist is generated which has an identical tracklist to one which already exists in the database, the time it was generated is updated to the current time, but it's not duplicated in the database, and no playlist is evicted. This allows users to go back and look at historical playlists.

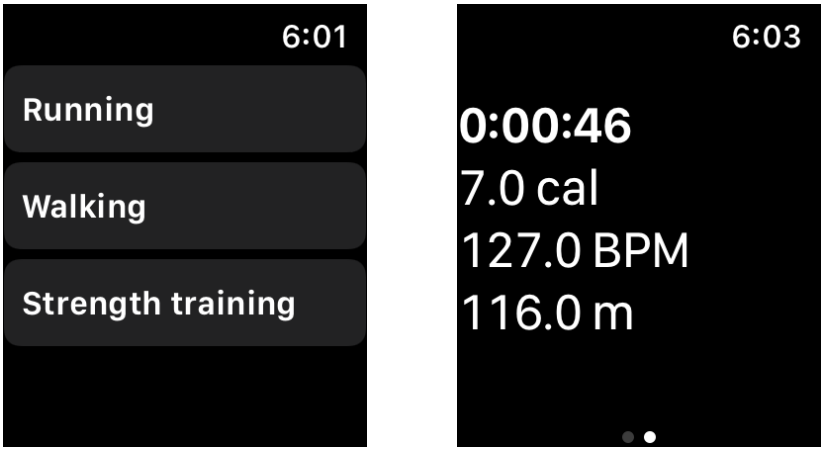
Having access to historical playlists is important, since over time ,as more data is collected and more songs enter the system, it's possible that generated playlists could change as more ideal songs become available. Also, since users can change the target values for their workouts, they can generate multiple playlists. One use case for this may be to generate a playlist for a 5km run and another for a 10km run, which both have appropriate workout feature values.

3.5 Real-time Track Recommendations

The next challenge was to adapt the project so that it could recommend the ideal track to a user in real-time. This was implemented in the form of an Apple Watch application, written in Swift. The foundation of the application was based on a watchOS tutorial by Apple called SpeedySloth [32], but integrated the Osti recommendation system to use the heart rate, calorie and distance data on device to alter the recommendations and play the appropriate songs to users in real time.

3.5.1 Initial Data Collection

The first state of the application once the user opens the application is to enter their user ID, and then select which type of workout they are completing. This is displayed in Figure 3.7a.



(a) User selects appropriate workout type (b) User vitals shown during workout

Figure 3.7: Osti real-time WatchOS application screenshots

Value	Description
Playlist	An ordered list of track identifiers describing the most recent suggested playlist for the given user-workout pair
User Targets	The target values of heart rate, total calories to burn, and total distance to cover
Recommendations	An ordered list of the top 200 tracks recommended for this user-workout pair
Track Data Map	A map of track information, such as duration and tempo, indexed by track identifier
Track Deltas	A map of each track's song deltas, indexed by the two/one/zero in common system described in Section 3.4.6

Table 3.5: Initial data transferred to Osti real-time application before a workout is started

Once this has been selected and a workout starts, the application sends a POST request to the `osti-recommender` Heroku machine, as described in Section 3.3, to the endpoint `get_initial_data`. This returns the values described in Table 3.5. This data is then transferred to internal data structures in Swift, to be used throughout the lifetime of the application.

3.5.2 Initial Tracklist

The initial data retrieved includes the suggested playlist, which is used as the foundation for the recommendations of the real-time application. These tracks are then played to the user using their Spotify account.

The endpoints which control a user's Spotify playback are all contained on the Javascript machine which is hosting the webapp, since this webapp already had integration with Spotify due to the playlist saving feature described later in Section 3.6.4. These endpoints are documented in Table 3.6.

To play the appropriate initial tracks for the user, either the `/playTracks` or `/playPlaylist` endpoint is used. The `/playPlaylist` is preferred, since it gives a nicer experience for the user, showing them where the playlist is coming from. However, if a user has not generated a playlist on Spotify for this particular workout, then the tracks are individually queued using the `/playTracks` endpoint instead. They both result in the initial tracklist being played to the user on Spotify.

Endpoint	Method	Parameters	Description
`/currentlyPlaying`	POST	uid - User ID	Returns the Osti Track ID of the currently playing song for the given user, null if nothing is playing, or song is unknown
`/playPlaylist`	POST	uid - User ID pid - Playlist ID	Starts playing the given playlist to the given user
`/playTracks`	POST	uid - User ID tids - List of Track IDs	Fills the queue with the given tracks and starts playing them for the given user

Table 3.6: Spotify control endpoints

3.5.3 Calculating Ideal Tracks

Once the initial tracklist has started playing, then the application needs to start analysing how a user is progressing with the workout, and update the music playing in accordance with this. The first stage of doing this is to calculate which are the ideal tracks out of the set of recommendations at the current moment in time for the user. This is calculated every 10 seconds, and the results will determine whether the music will change.

To calculate the ideal tracks, the application uses a similar method to the playlist generation described in Section 3.4.6. However, instead of using the workout deltas, which describe the average user workout at a certain time, we use the real-time user deltas. Due to limitations of the HealthKit API, only three of five of the metrics used for playlist generation can be used in real-time; heart rate, distance travelled and calories burned.

The heart rate delta is calculated in the following manner:

$$(\delta_w^u)_{hr} = \text{Target Heart Rate} - \text{Current Heart Rate}$$

The calorie delta, which is how many calories should be being burned every 10 seconds from now on, is calculated using the following equation:

$$(\delta_w^u)_{cal} = \frac{\text{Total Target Calories} + \text{Calorie Deficit}}{\text{Target Length in Minutes} \times 6}$$

Where Calorie Deficit =

$$(\frac{\text{Elapsed Time in Seconds}}{\text{Target Time in Seconds}} \times \text{Total Target Calories}) - \text{Calories Burned So Far}$$

The distance delta, which is how far should be travelled every 10 seconds from now on, is calculated similarly to calorie delta, using the following equation:

$$(\delta_w^u)_{dist} = \frac{\text{Target Distance} + \text{Distance Deficit}}{\text{Target Length in Minutes} \times 6}$$

Where Distance Deficit =

$$(\frac{\text{Elapsed Time in Seconds}}{\text{Target Time in Seconds}} \times \text{Target Distance}) - \text{Distance Travelled So Far}$$

These deltas are then compared to each song delta, similarly to Section 3.4.6, and the cosine distance between each track and the deltas are calculated. This is repeated three times, once for song deltas for the same user-workout pair, once for song deltas with one of either the user or

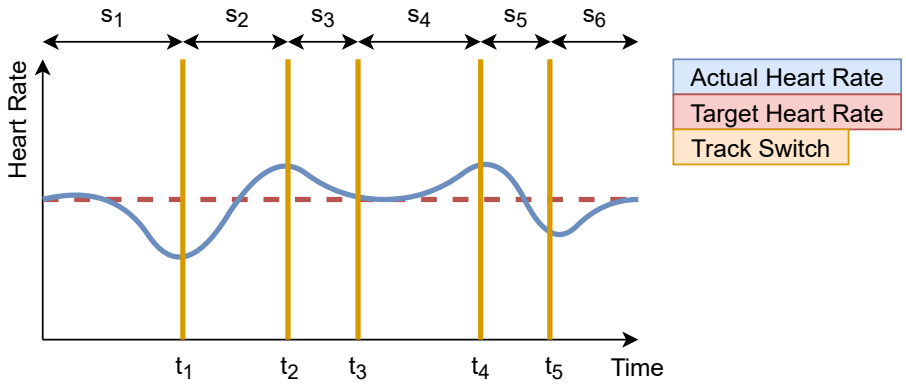


Figure 3.8: Real-Time Track Switching Example

workout the same, and one for song deltas with neither the same. The cosine distances for these are then weighted using the ratio 5:2:1.

Once this has been calculated for a track, then the tempo distance and position in the recommendation list are also taken into account, using the same equations as in Section 3.4.6.

3.5.4 Switching Tracks

The previous section has many similarities to playlist generation. However, the next step is different. Instead of just picking the track which is most ideal for the current position, the algorithm has to make a decision to either keep playing the current track, or skip to a more ideal one.

The application first gets the currently playing track using the `/currentlyPlaying` endpoint. If this song is in the top 10 ideal songs calculated in the previous section, then nothing happens, and the track keeps playing. However, if it is not ideal, then a counter *wrongCount* is initialised and incremented by one. Once this counter exceeds 3, which means that the track has not been ideal for 30 consecutive seconds, it's decided that the song playing is not ideal, and needs to be changed to adjust the user's activity to a more appropriate range. First, the counter is reset, and then the application takes the top recommended tracks and repeatedly adds the top track from this to a queue, until the length of the queue is at least as long as the length of the target workout.

This queue of songs is then played to the user on Spotify, using the `playTracks` endpoint. Since these songs are the most likely to bring their heart rate, calorie burn rate and distance covered closer to the target values, the user should return closer to their target. Once this happens, the ideal songs will change again to try and keep their workout performance stable. This process will repeat until the length of the duration of the workout.

Figure 3.8 illustrates an example of this using heart rate. Initially, song s_1 is playing, which would be equivalent of a song from the initial playlist. However, the user's heart rate is declining, and is below the target heart rate. Therefore, at time point t_1, the song is switched to s_2, which is a song that is known to raise users' heart rates. As can be seen between time points t_1 and t_2, the user's heart rate rises. However, since it starts to overshoot the target, at time point t_2, the song is switched again in order to bring the heart rate back down. Once the heart rate is stable, at time point t_3, the track is once again switched to one which keeps the heart rate stable. This process can then be seen to happen again in reverse order,

3.6 User Interface

In order to create a working system for users to test, a user interface needed to be created. The main features needed were initial setup, viewing recommendations, providing feedback on recommendations, viewing and adjusting workout targets and viewing and downloading playlists.

The interface was built as a web application, hosted at `https://osti.uk`, using a React framework called Next.js [33].

3.6.1 Initial Setup

Since this application runs using real user data, the first key use for a user interface is for connecting the Last.FM, Google Fit and Spotify APIs. To link these together for each user, and overall user account is required, which is managed by a single Google login by the user. Once an account is created, the `/profile` page hosts the ability to authorise access to their LastFM and Google Fit accounts, giving appropriate permissions for each. Once this is completed, they are then prompted to trigger the start of the data collection pipeline detailed in section 3.2.

3.6.2 Recommendations and User Feedback

The second use of the web interface is the ability for a user to view which songs will be recommended to them for each potential workout. This is a visualisation of the results of the Music Recommendation Engine described in Section 3.3. For a logged in user, they can navigate to the `/recommendations` page, which lists all of the workouts which they have recommendations for.

From here, they can select a workout and be taken to a more in-depth page which contains the ability to customise the targets for this workout type (Section 3.6.3), playlists generated for this workout type (Section 3.6.4), the top 200 recommended songs for this workout type and up to the top 100 most listened to songs by them for this workout type.

The two lists of tracks on this page allow the user to see both which tracks the Music Recommendation Engine thinks they would most like listening to in the context of this workout type, along with what historically they've listened to the most during this workout type. They display basic track information, such as name, artist and cover artwork, as well as a 'boost' drop-down, which allows the user to give feedback to the track in this context as detailed in Section 3.3.5. Once a user updates the boost value, this is instantly stored in the database, and then when the Music Recommendation System is next run, this boost value will be taken into account, so the track in the corresponding context will either be moved up or down the ranking. An example of this page for a running workout can be seen below in Figure 3.9.

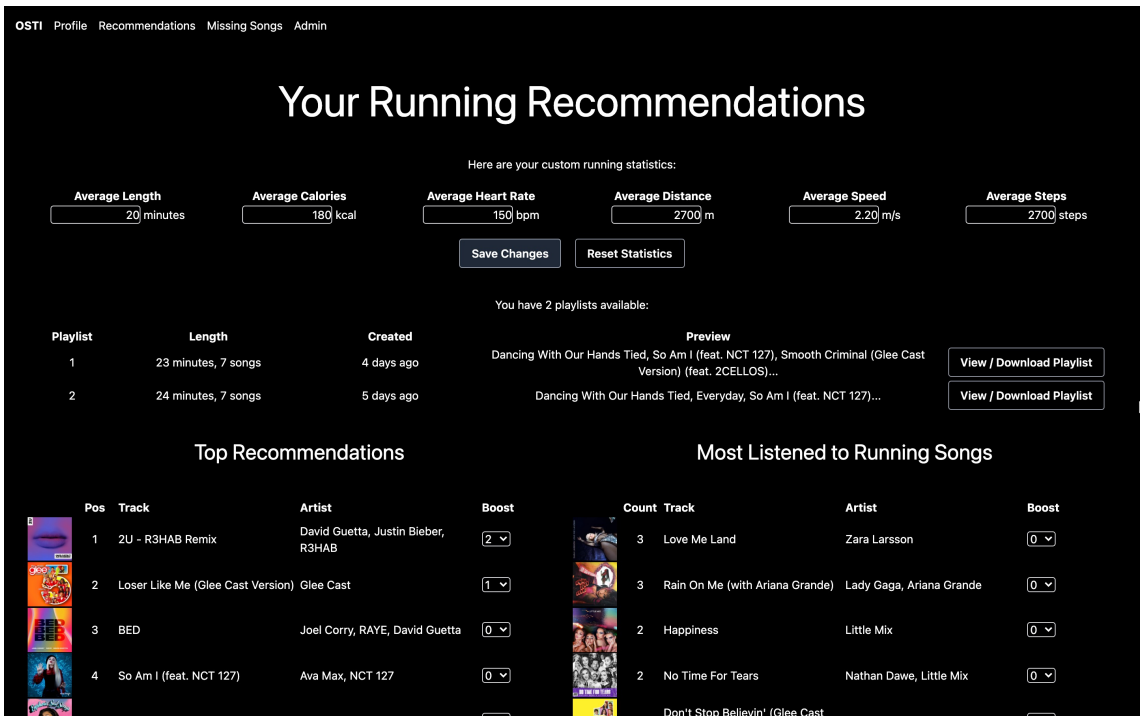


Figure 3.9: An example of the Recommendation page on the Osti web interface for a running workout

3.6.3 User Workout Targets

Also on each individual workout recommendation page is the interface which allows a user to view and alter the targets for each of the workout metrics, such as average heart rate and total calories burned. The boxes are populated with the default values, so that a user can see what their 'average workout' looks like, and once they update and save these values, any further playlists which are generated will use these as baseline targets. The ability to reset these to the default values is also available, since a user may accidentally update these. This can be seen in Figure 3.10.

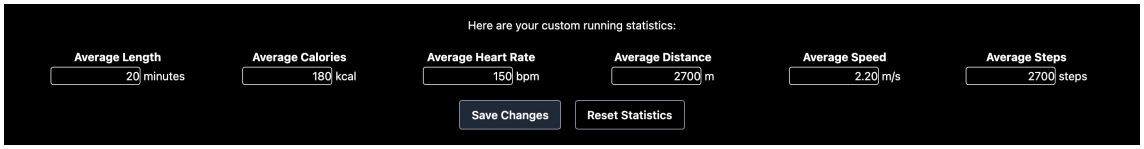


Figure 3.10: An example of the User Workout Targets for a running workout on the Recommendations page of the Osti web interface

3.6.4 Generated Playlists

The final section of the workout recommendation pages are the generated playlists. As described in Section 3.4.6, the 10 most recent unique playlists which have been generated are stored for each user-workout pair, and are displayed to the user. They're displayed along with their length in minutes, the number of songs they contain, when they were generated, and a preview of the tracks. This can be seen in Figure 3.11 They also have a link to a playlist-specific page.

On the playlist pages, the user has the option to save the playlist to their Spotify library, and if it's already saved, to open it up directly in the application. This allows quick access to start playing the playlist at the beginning of a workout. The full tracklist, along with track info such as artist and track tempo can also be seen on this page, as seen in Figure 3.12.

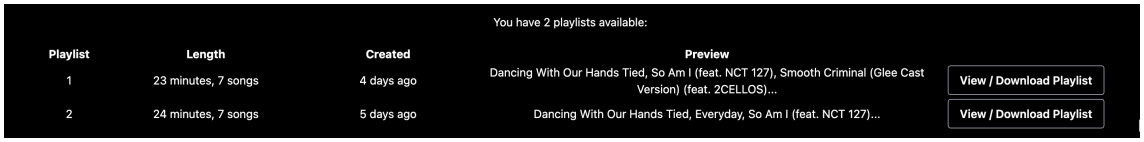


Figure 3.11: An example of the most recently generated playlists for a running workout on the Recommendations page of the Osti web interface

3.6.5 Missing Songs

The next page, located at `/missing`, is where any tracks which the fetcher has not been able to find on Spotify are collected. Since there is not a one-to-one link between Last.FM scrobbles and Spotify tracks, it relies on the search endpoint of the Spotify API, as detailed in Section 3.2.2, and therefore for tracks which cannot be found with this endpoint, features cannot be retrieved, and so they cannot be recommended.

Therefore to combat this issue, the page lists tracks which the user has listened to that have not been located on Spotify, ordered by how many times they have listened to them. It then has a text box, where the Spotify URL, which can be easily obtained from the Spotify application, can be pasted in. This will then link the track in Osti's database to the correct track on Spotify, and complete the feature collection for this song. From then on, this song will be available in recommendations to all users. It is rare that a user will need to do this often, however if the unfortunate circumstance happened where a song that a user listened to regularly cannot be located, this gives them the ability to incorporate it into the system. This page can be seen in Figure 3.13.

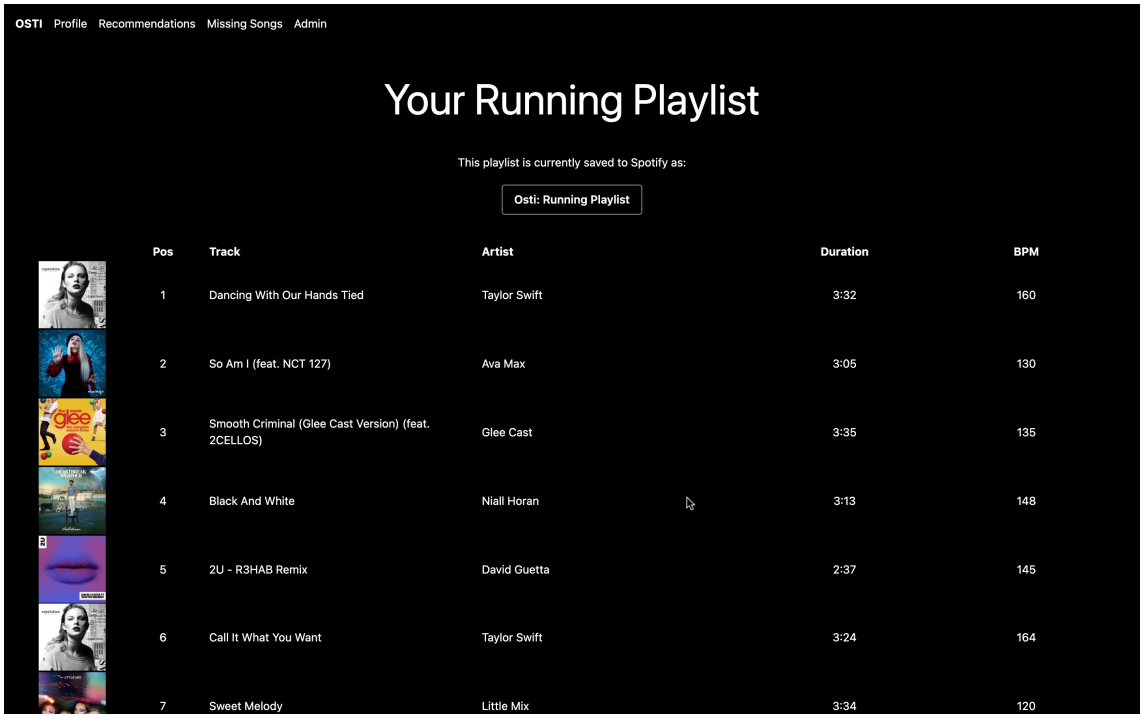


Figure 3.12: An example of the page which displays the tracks in a playlist, and allows the user to download it to Spotify and then open the playlist, on the Osti web interface

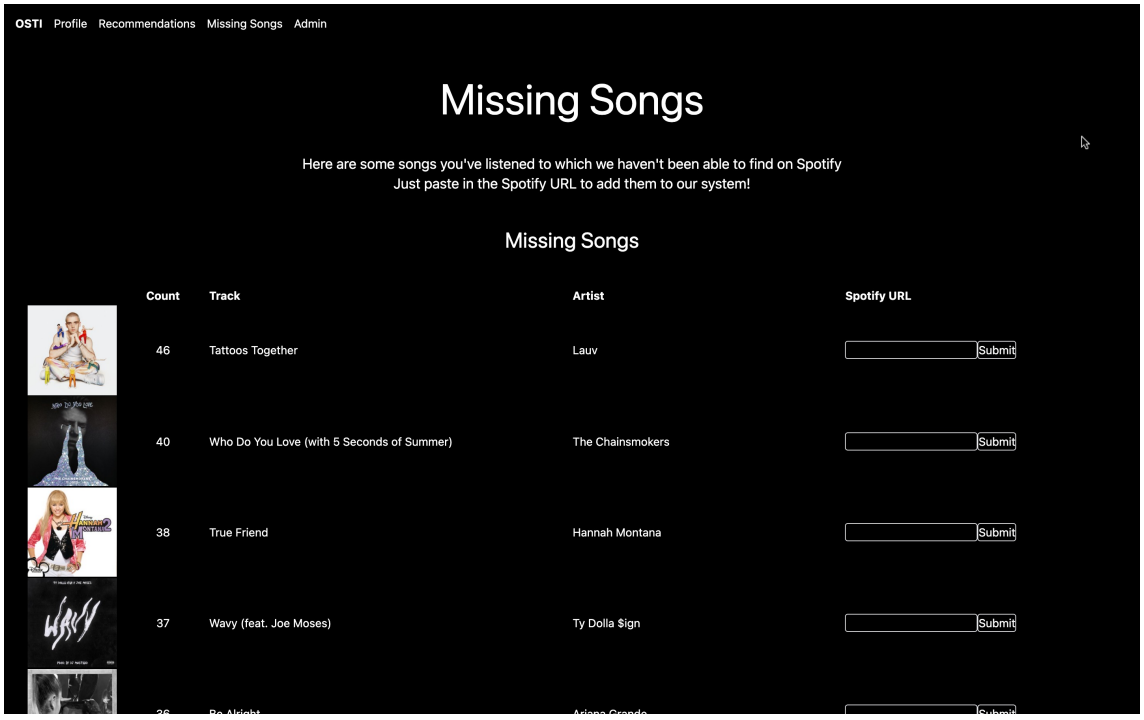


Figure 3.13: An example of the missing songs page on the Osti web interface

Chapter 4

Evaluation

This chapter evaluates how well the project has met the objectives set out in Section 1.2, using both data visualisation (see Section 4.1) and user testing (see Section 4.2).

4.1 Data Visualisation

Since recommendations are subjective, it's hard to quantify a metric to score how accurate they are without asking the listened for their opinion (as is done in Section 4.2.1). Therefore, a useful way to see how the Music Recommendation Engine functions is by visualising the data.

4.1.1 Feature Analysis

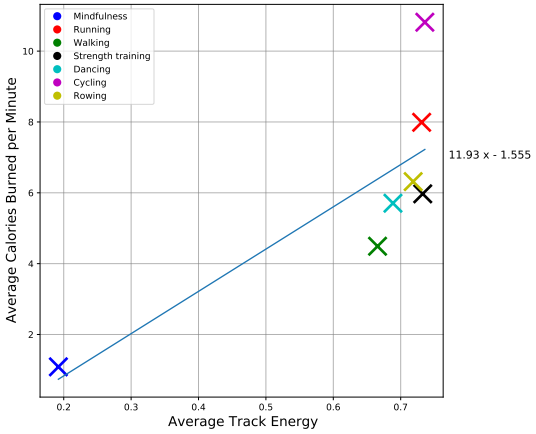
The first method of doing this is to analyse some of the features of tracks described in Table 3.1 for different workout types, for both listens recorded from users, as well as the recommendations given by the system.

Figure 4.1 shows how the average track energy vs average calories burned per minute varies across workout types, both for tracks users have listened to (left) and the recommendations given (right). Both of these graphs have a positive correlation, implying that there is a correlation between listening to higher energy tracks and burning more calories, as would be expected. The music recommendation engine has also followed this trend when recommending tracks, showing that the recommendations are suitable based on this track feature. We can see that in both cases, a calming activity such as Mindfulness burns a lot less calories, and is associated with tracks of much lower energy, and the opposite is shown for the higher energy activities such as Cycling.

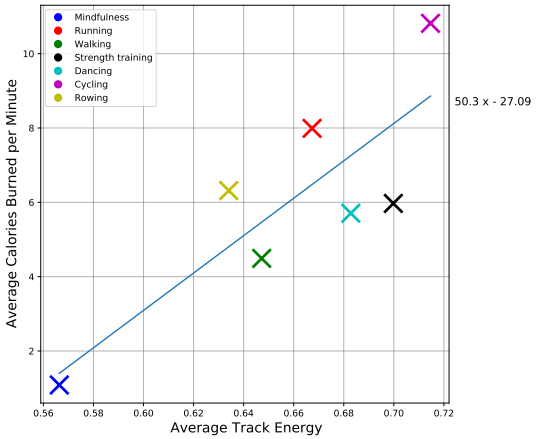
However, one difference between the recorded tracks and recommendations is the range of track energy values. For the recorded tracks, the energy is pretty similar for all workout types, however is significantly lower in the Mindfulness context. The average track energy for Mindfuless recommendations is still lower, however is much closer to the other activity types than in the recorded tracks. This implies that the change in track energy between workout types is not as significant in the recommendations. One explanation may be that the recommendations used here were only for one user, since only one user had completed a workout of this type, whereas the recorded tracks use the dataset described in Section 3.3.3, and therefore has recorded tracks from 10 different users.

Figure 4.2 shows how average track acousticness vs average calories burned per minute varies across workout types, again for recorded tracks and recommended tracks. We can see the correlation here is negative, implying that the more calories that are burned on average per minute, the less acoustic a track will be. This aligns with expectations, since acousticness is negatively correlated to energy [34].

Figure 4.3 shows a strong positive correlation between danceability and calories burned per minute in the recommended tracks. This correlation is less strong for the recorded tracks, however a positive correlation can still be seen. One interesting point is the workout type of Dancing

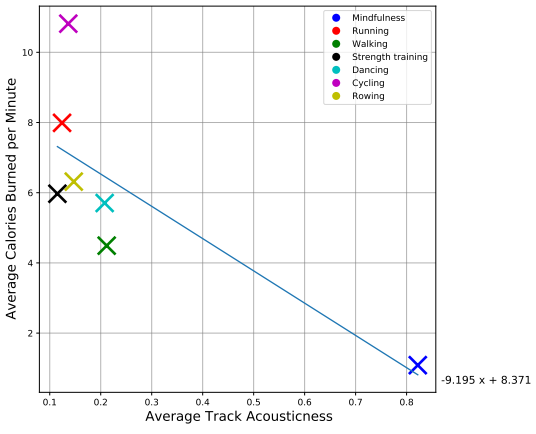


(a) Tracks listened to by users in given workout context

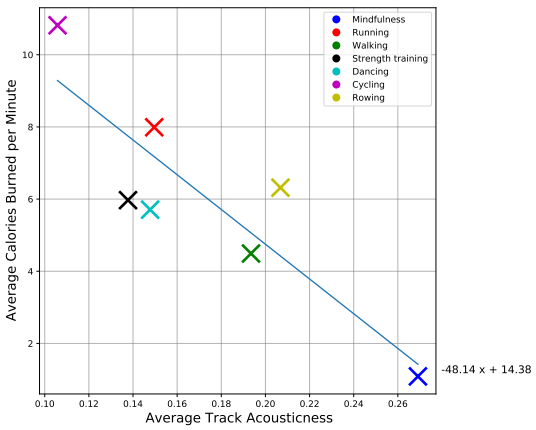


(b) Recommended Tracks for given workout context

Figure 4.1: Average Track Energy vs Average Calories Burned per Minute



(a) Tracks listened to by users in given workout context



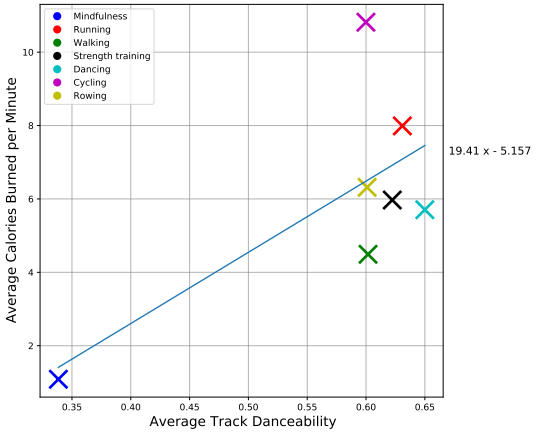
(b) Recommended Tracks for given workout context

Figure 4.2: Average Track Acousticness vs Average Calories Burned per Minute

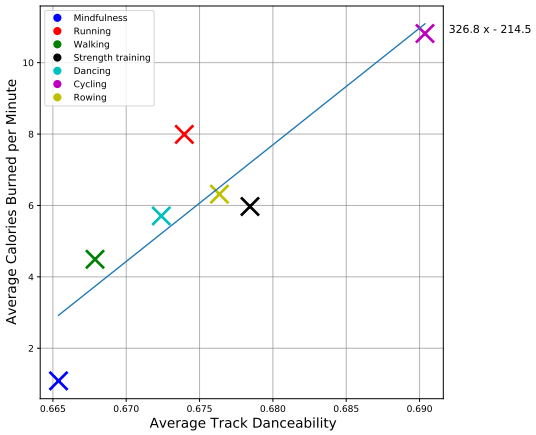
records the highest danceability score in the recorded tracks, as is expected, however only the 5th highest danceability score in the recommended tracks. This may be due to the limited number of tracks used for recommendations, at 200 tracks for this graph due to the small number of users who have completed this type of workout, whereas 26,029 tracks were recorded for this workout track using the data from the earlier described Spotify Million Playlist Dataset. Therefore, this may change as more users enter the system. It's also been shows danceability and energy do not directly correlate [34], and therefore this shows this relationship has been observed and learned by the music recommendation engine independently of the energy feature.

Figure 4.4 shows a similar trend with loudness. It's been shown that loudness and energy are highly correlated [34], and, when also looking at Figure 4.1, we can see this trend continue in both the observed values as well as our recommendations. Figures 4.1a and 4.4a reinforce this strong correlation, since the different workout types have similar placement on both graphs. The recommendation graphs (Figures 4.1b and 4.4b) also show a positive correlation, albeit weaker. However, this may be due to the limited nubmer of users in the system.

The final feature considered is instrumentalness, in Figure 4.5. As can be clearly seen in the graphs, the correlation between instrumentalness of tracks and calories burned is not the same for observed tracks and recommended tracks. Figure 4.5a is more expected, since the workouts which burn more calories are less instrumental, and therefore have more lyrics. Tracks with lyrics have

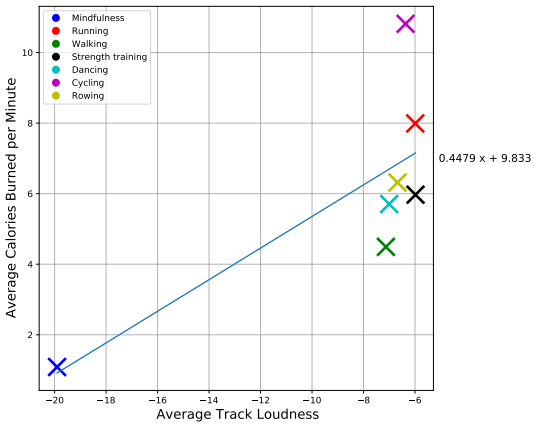


(a) Tracks listened to by users in given workout context

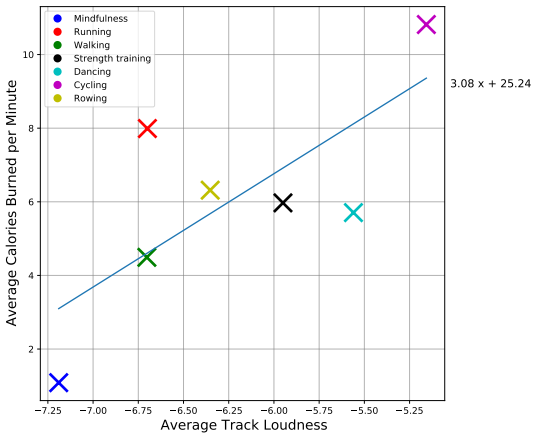


(b) Recommended Tracks for given workout context

Figure 4.3: Average Track Danceability vs Average Calories Burned per Minute



(a) Tracks listened to by users in given workout context



(b) Recommended Tracks for given workout context

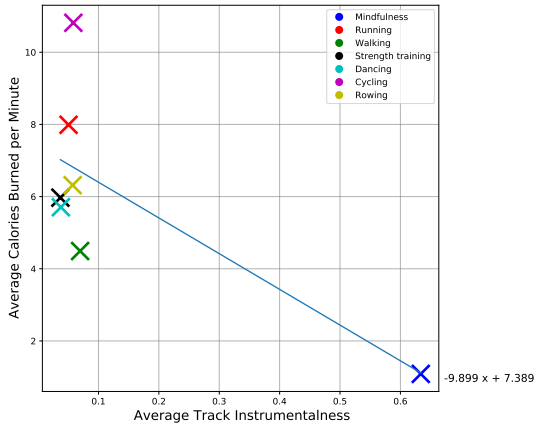
Figure 4.4: Average Track Loudness vs Average Calories Burned per Minute

been shown to be more motivational for exercise than tracks without [35], and so this follows the trend shown in our observations. However, this trend is not replicated in out recommendations. One key outlier is the mindfulness activity type. As we can see from Figure 4.2b, Osti recommends more acoustic songs for mindfulness playlists than instrumental songs. This may be a user preference, since we know users have different preferences for different contexts [16], and therefore with more users this could be investigated further.

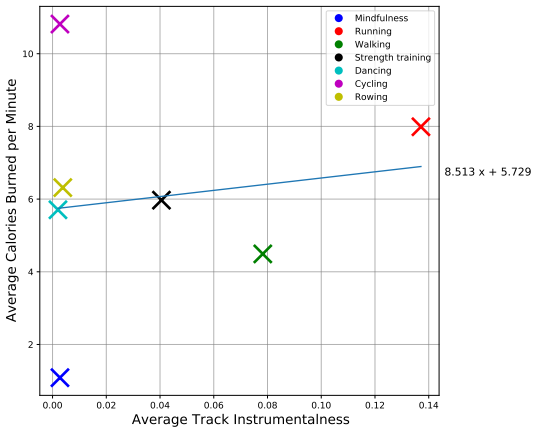
4.1.2 Known vs Unknown Recommended Tracks

The next data we wanted to visualise was the percentage of recommended tracks which were known to a user. The ability to recommend tracks a user hasn't listened to before is a key advantage of the Osti system, allowing it to generate recommendations based on minimal data using a hybrid approach of Collaborative Filtering and Content-Based recommendations. This means recommendations can be generated from any number of tracks, and will become more accurate over time as the number of tracks listened to in a particular context increases, giving a clearer image of the user's music taste for this particular context.

To visualise the amount of tracks being recommended to the user that are known, for each user-workout pair Osti has generated recommendations for, we plotted the number of tracks listened to by a user in that particular context against the percentage of tracks which are known to them



(a) Tracks listened to by users in given workout context



(b) Recommended Tracks for given workout context

Figure 4.5: Average Track Instrumentalness vs Average Calories Burned per Minute

in that context. This can be seen in Figure 4.6. As was expected, as the number of tracks a user has listened to in a particular context increases, so does the percentage of recommended tracks which are known in that context. This implies that the recommendations the system generates for the user-workout pair align with the user's music taste for this workout. However, the graph also shows that for the data we currently have, the majority of user-workout pairs have a fairly low total number of tracks listened to in that context - all but 2 of these pairs are below 100 tracks. Therefore, a longer study would be needed to confirm this trend in the long term.

All but one sets of recommendations are below 30% known songs, which shows that the music recommendation engine does not just suggest the top tracks for a user, and is using the methods of collaborative filtering and content-based recommendations to show tracks which are more ideal for the context. The appropriateness of these recommendations is explored further with user testing in Section 4.2.

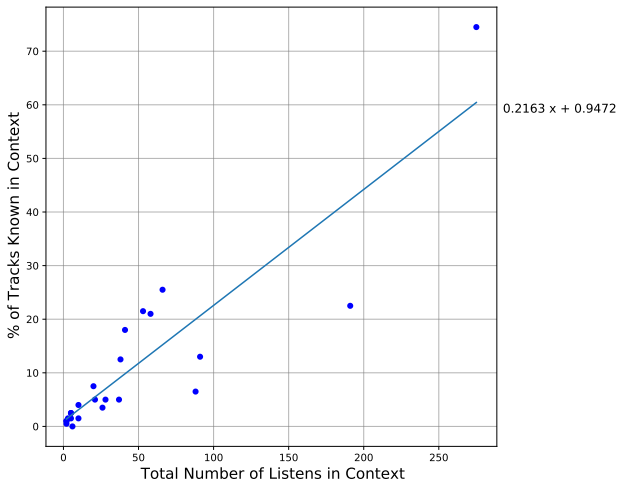


Figure 4.6: Number of known tracks in a given context relative to how many tracks a user has listened to in that context

4.2 User Testing

Since this is a user-facing application, user testing was an integral method of evaluating success. This application has two main methods of recommendation: playlists and real-time recommendations, which will both be evaluated.

4.2.1 Playlist Generation

One key metric for measuring the appropriateness of generated playlists is to compare these playlists against other combinations of songs from different sources. In this case, we will use four playlists per user-workout combination:

- Osti-Generated Playlist
- Spotify's Soundtrack Your Workout Playlist [23]
- User Hand-Curated Playlist
- Generic Workout Playlist (e.g. 'Fun Run' from Spotify [36])

These playlists were all be made to be the same length, and anonymised so the user is not biased towards one. 3 users participated in this study, who all came from varied backgrounds to ensure that the results are representative of an actual population. They are varied based on gender, age, ethnicity, fitness level and music taste.

We measured how close the user's heart rate, calories burned, distance covered, average speed and steps taken were to the targets which were set for the workout using standardised cosine similarity and standardised Euclidean distance. Standardisation is where you subtract the mean of each recorded vital from each, and then divide by the equivalent standard deviation, such that one vital does not influence the similarity more than another. Despite cosine distance not taking magnitude as a factor, it makes it easier to see the difference between the different values.

We did this for three workout types: running, walking and strength training. (For strength training, we only consider heart rate and calories burned, since this does not involve moving over a distance). The full results of these tests can be seen in Appendix B, however an overview of the results is discussed here.

4.2.1.1 Running

The first workout we look at is a Running workout. As can be seen in Table 4.1 and Table 4.2, the Osti playlist had the largest cosine simiarity and smallest Euclidian distance from the target values, and therefore kept the user closest to target. The generic running playlist also had similar success in keeping the user close to target.

Figure 4.7 shows how the user's speed changed over time, marked with when a each track started playing (other vitals, as well as a list of tracks played in each workout can be found in Appendix B). As we can see, with the Osti playlist and Generic playlist, there is a clear change in speed when the song changes, however moreso with the Osti playlist. This implies that the tracks encourage the user to run at a specific speed. However, since Osti knows this target speed (in this case 2.2m/s), it can find songs which will keep a user around this average. It can be seen that it has done this by alternating between tracks which encourage the user to run at higher and lower speeds.

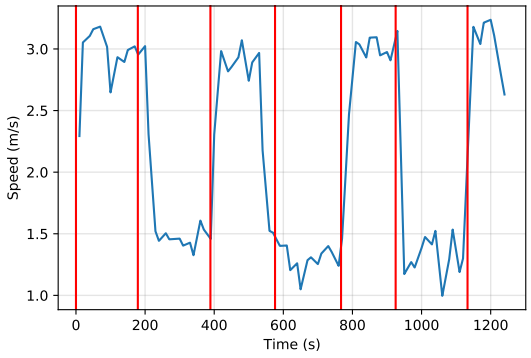
Playlist Type	**Standardised Cosine Similarity**	**Standardised Euclidian Distance**
Osti-Generated Playlist	0.6877171843	1.011975091
Spotify's Soundtrack Your Workout Playlist [23]	-0.39378341	2.42696652
User Hand-Curated Playlist	-0.03869123708	1.374357225
Generic Workout Playlist ('Fun Run' from Spotify [36])	0.3005029216	1.068987713

Table 4.1: Standardised cosine distance between target running vitals and observed running vitals for different playlist types

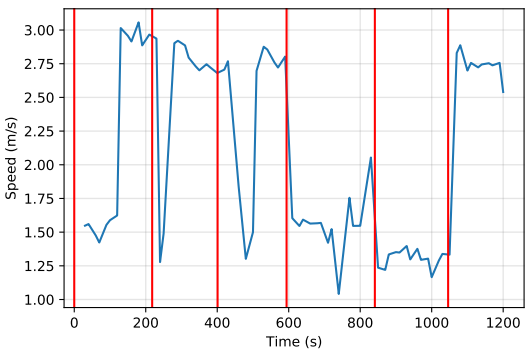
Spotify's Soundtrack Your Workout's playlist can be seen to encourage a user to run at a faster speed for the first 3 tracks. However, by the fourth track they were probably tired, and this caused their speed to drop significantly. Track 5 was also not known to the user, and therefore didn't have the impact to increase their speed again, causing under-performance which can be seen in Table 4.2.

Playlist Type	**Duration (mins)**	**Average Heart Rate (bpm)**	**Calories Burned (kcal)**	**Distance Covered (m)**	**Average Speed (m/s)**	**Steps Taken**
Target	**20**	**150**	**230**	**2700**	**2.20**	**2700**
Osti-Generated Playlist	21.1	149.9	229.7	2713.7	2.17	2826
Spotify's Soundtrack Your Workout Playlist [23]	20.3	146.6	188.0	2357.2	2.08	2521
User Hand-Curated Playlist	20.5	163.3	238.4	2686.3	2.28	2758
Generic Workout Playlist ("Fun Run" from Spotify [36])	21.2	155.9	238.3	2655.7	2.23	2651

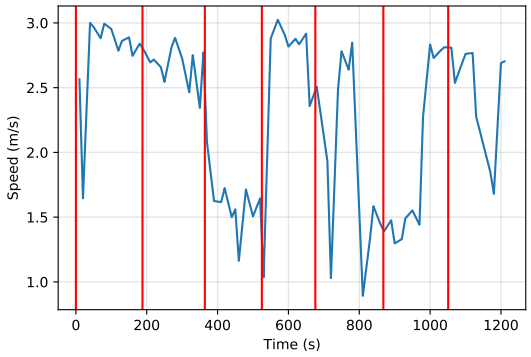
Table 4.2: Comparison of user running vitals for different playlist types



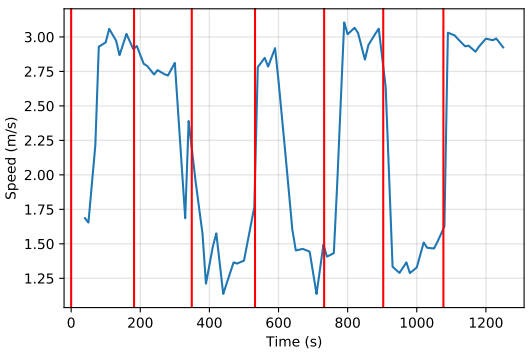
(a) Osti Playlist



(b) Spotify's Soundtrack Your Workout Playlist



(c) User Hand-Curated Playlist



(d) Generic Workout Playlist

Figure 4.7: Time vs Speed for a Running Workout

4.2.1.2 Walking

The next workout type to be tested was a walking workout. This has a lower intensity than the running workout, and therefore we wanted to see whether this made it easier to more difficult to encourage the user to reach their target vitals with music.

As can be seen in Table 4.3, the Osti Playlist again had the highest cosine similarity and lowest Euclidean distance. However, in this case the User-Curated playlist also had a very high cosine similarity and low Euclidean distance, implying that it also kept the user very close to their target vitals.

Figure 4.8 shows how a user's heart rate changed over time, along with when certain tracks were played (other vitals, as well as a list of tracks played in each workout can be found in Appendix B). This figure shows that the Osti playlist caused a much smoother change in heart rate over time, whereas other playlists, such as Spotify's Sountrack Your Workout playlist, caused the heart rate to fluctuate a lot more.

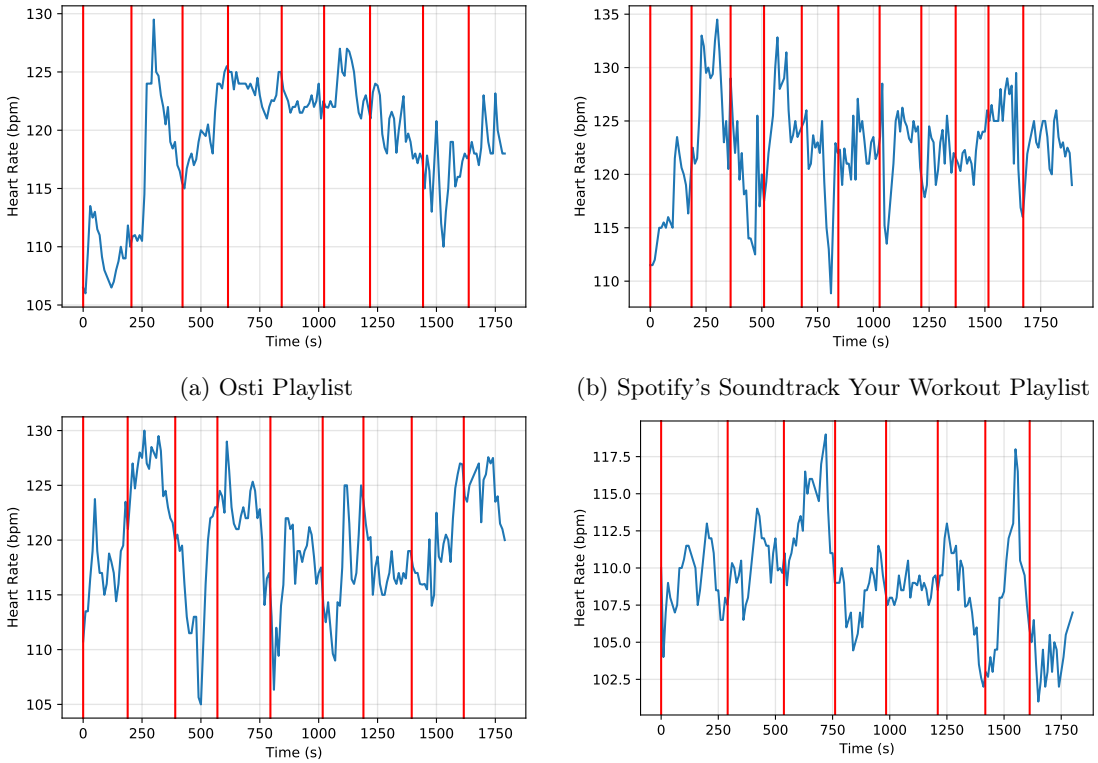
We can also see that the Generic playlist workout, which had an average heart rate of 109.0 bpm, never even caused the user to reach the target heart rate of 120 bpm. Since this playlist consisted of only one track out of seven which was known to the user, it implies that the user was less motivated to walk at a faster pace, thus increasing their heart rate, since they didn't know the music playing. This is also reflected in the lower average speed of 1.37 m/s.

Playlist Type	**Standardised Cosine Similarity**	**Standardised Euclidian Distance**
Osti-Generated Playlist	0.819692162	2.069791088
Spotify's Soundtrack Your Workout Playlist [23]	-0.464170922	3.464220045
User Hand-Curated Playlist	0.6491089589	2.429297181
Generic Workout Playlist ('Walking Music' from Spotify [37])	-0.3155987036	3.524149311

Table 4.3: Standardised cosine distance between target walking vitals and observed running vitals for different playlist types

Playlist Type	**Duration (mins)**	**Average Heart Rate (bpm)**	**Calories Burned (kcal)**	**Distance Covered (m)**	**Average Speed (m/s)**	**Steps Taken**
Target	**30**	**120**	**111**	**2500**	**1.44**	**3000**
Osti-Generated Playlist	29.9	119.2	119.0	2521.8	1.41	3235
Spotify's Soundtrack Your Workout Playlist [23]	31.5	122.4	140.1	2608.3	1.41	3370
User Hand-Curated Playlist	30.0	119.7	129.2	2532.6	1.42	2988
Generic Workout Playlist ('Walking Music' from Spotify [37])	30.1	109.0	119.0	2460.4	1.37	3376

Table 4.4: Comparison of user walking vitals for different playlist types



(a) Osti Playlist

(b) Spotify's Soundtrack Your Workout Playlist

(c) User Hand-Curated Playlist

(d) Generic Workout Playlist

Figure 4.8: Time vs Heart Rate for a Walking Workout

4.2.1.3 Strength Training

The final workout type tested was a strength training workout. This is different from the previous two tests since it does not involve moving over a distance, and therefore the only metrics which could be collected are heart rate and calories burned.

Table 4.5 shows that the user-curated playlist performed the best according to cosine distance, and the Osti playlists performed the best according to Euclidean distance.

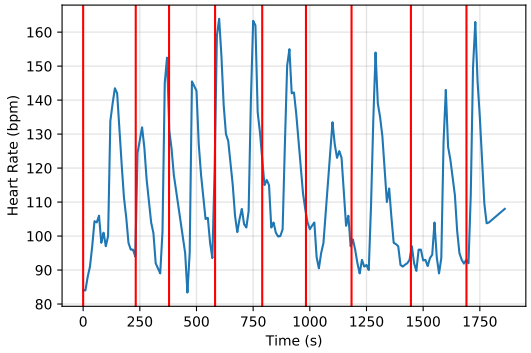
However, Figure 4.9 shows that there is not much of a correlation between heart rate and when a track changes for any playlist type. This is probably because of the different nature of this workout type: a user will usually do sets of weight lifting to train strength, and therefore it's harder to visualise the effect of the tracks as physical exertion is not constant, as it is with running and walking workouts. This implies that it's harder to align music with exact vital changes, however the results shown in Table 4.5 show that, probably using the vital of calories burned, since it's cumulative over time, Osti has managed to predict a playlist which kept the user close to target. The user's hand curated playlist also managed to do this.

Playlist Type	**Standardised Cosine Similarity**	**Standardised Euclidian Distance**
Osti-Generated Playlist	0.7943001801	1.466018665
Spotify's Soundtrack Your Workout Playlist [23]	-0.5096565764	3.551569157
User Hand-Curated Playlist	0.8560799298	2.233827903
Generic Workout Playlist ('Workout' from Spotify [38])	-0.6341866047	2.913684112

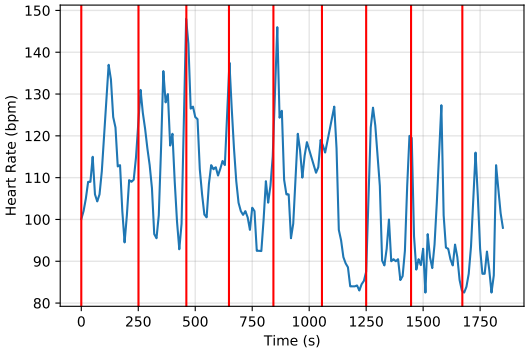
Table 4.5: Standardised cosine distance between target strength training vitals and observed running vitals for different playlist types

Playlist Type	Duration (mins)	Average Heart Rate (bpm)	Calories Burned (kcal)
Target	**30**	**115**	**170**
Osti-Generated Playlist	31.0	112.4	183.5
Spotify's Soundtrack Your Workout Playlist [23]	31.0	106.4	155.4
User Hand-Curated Playlist	31.8	109.3	171.9
Generic Workout Playlist ('Workout' from Spotify [38])	31.8	107.9	181.3

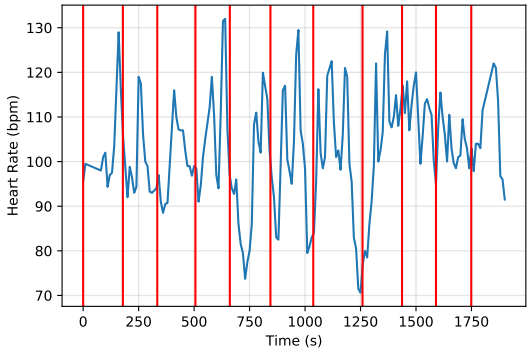
Table 4.6: Comparison of user strength training vitals for different playlist types



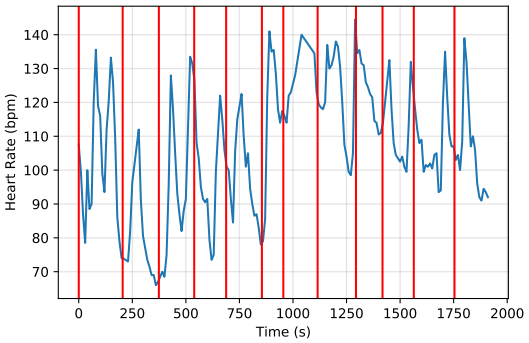
(a) Osti Playlist



(b) Spotify's Soundtrack Your Workout Playlist



(c) User Hand-Curated Playlist



(d) Generic Workout Playlist

Figure 4.9: Time vs Heart Rate for a Strength Training Workout

4.2.1.4 Overview

As we can see in Figure 4.10, Osti has the standardised cosine similarity closest to 1 for all workout types. This shows that it was the playlist which generated recommendations which helped the user reach their target values the closest. The user-generated playlists also had a very high or higher cosine similarity. This makes sense, since these playlists have been created by the user to help them obtain target workout performance, and therefore are likely to have the desired effect, since the user knows their music preferences better than an algorithm can. The outlier here is the user-generated running playlist. This caused the user to over-exert themselves and be above target. This may be viewed as successful, however the target of this experiment was to keep the user as close as possible to their targets, since this prevents over-exertion. It's possible that the user didn't curate this playlist correctly, and therefore further testing could be done to verify this result.

Figure 4.11 shows the Euclidean distances, grouped by workout type. We can see that the average Euclidean distance was lower for the running workout type, implying that the playlists all kept the user closer to their target values. The Osti playlists had the smallest Euclidean distance

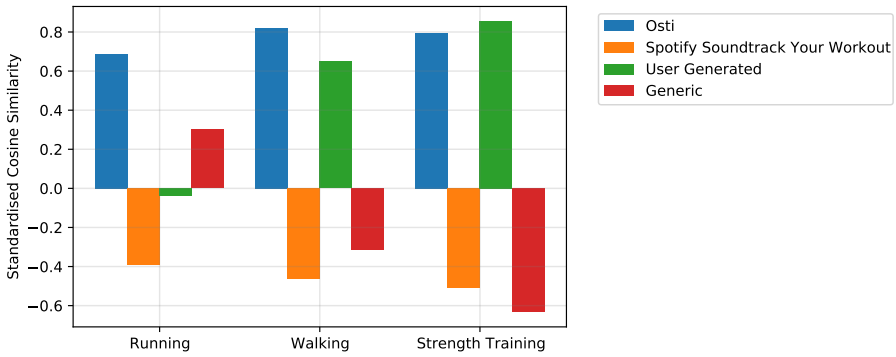


Figure 4.10: Standardised Cosine Similarity by Workout and Playlist Type

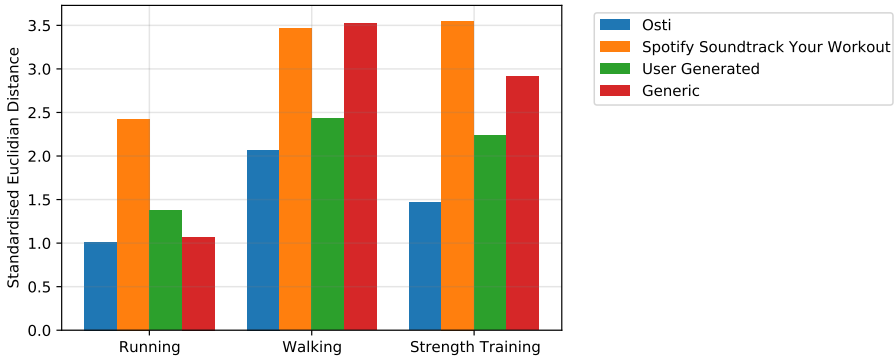


Figure 4.11: Standardised Euclidean Distance by Workout and Playlist Type

for each workout type, implying that it kept the user closest to target for all workout types. However, as discussed above, it's harder to justify this for workouts without constant exertion, such as strength training, since these vitals fluctuate regularly due to factors external to the user's exertion.

Figure 4.12: Photos of users completing user tests

4.2.2 Real-time Recommendations

For real time recommendations, we decided to similarly test how close the user's heart rate, distance travelled and calories burned were to the target, and whether the changing of songs in real time actually had an impact on these metrics. We did this on two workout types: running and strength training, since these vary in intensity.

A list of which tracks were listened to for each workout, along with the vital graphs for each workout, can be found in Appendix C.

4.2.2.1 Running

The first workout type we considered was a running workout. This was completed by the same user who completed the running workouts in the previous section. Since running usually has a more stable exertion, where vitals don't fluctuate quickly, we wanted to see how often the Osti algorithm had to skip songs. Figure 4.13 shows user vitals over the course of the workout. The red vertical lines represent a track being played naturally (either initially or after the previous one finished playing. Yellow vertical lines represent a track being played due to the previous one being skipped. The green horizontal line shows the target for that vital throughout the workout.

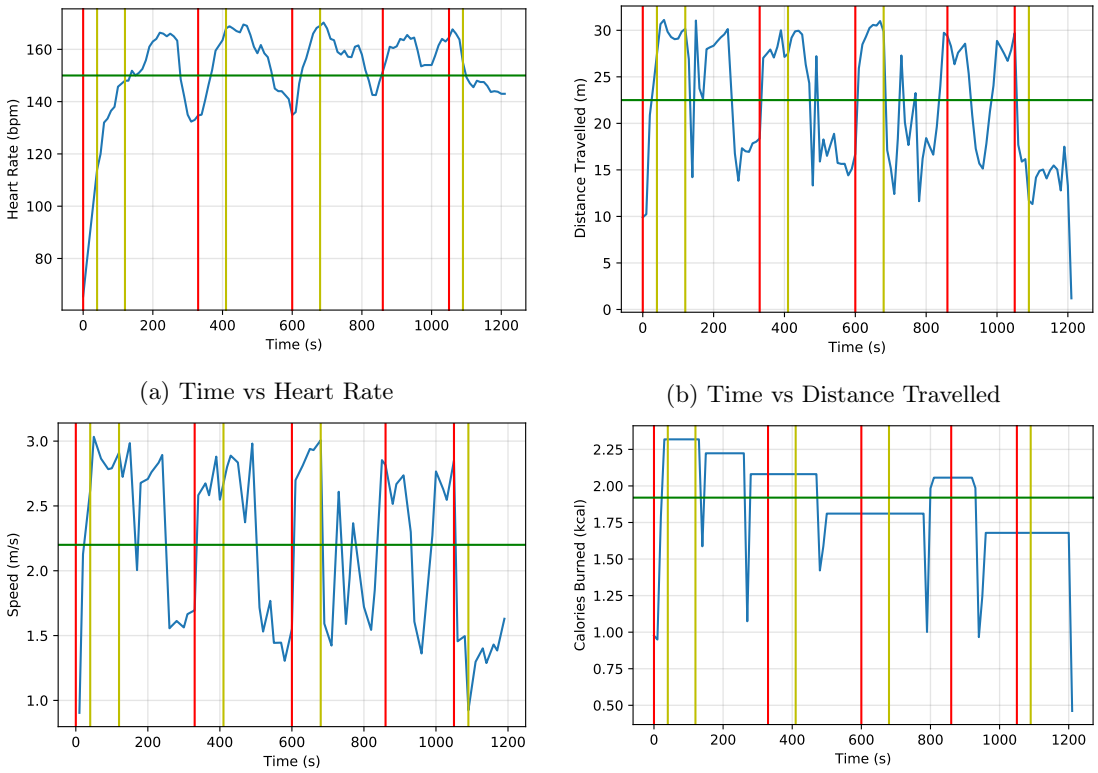
As we can see from Figure 4.13a, the changes in track usually occur when the user's heart rate is at a maximum or minimum, suggesting that the change in track results in a change in direction of heart rate. Similar patterns can be seen with speed and distance travelled, suggesting that the user speeds up or slows down in order to go towards the target viral. This suggests that the suggestions were appropriate in helping the user stay on track.

We can see from Figure 4.13 that 5 out of the 10 tracks played were eventually skipped, in order to replace them with a more appropriate track, amounting in a 50% skip rate. After speaking to the user, they said "The points at which tracks skipped aligned with when I was over- or under-exerting myself, and were not too frequent to become annoying". This, along with the results from Figure 4.13 suggest that tracks were only skipped when a user was staying above or below target for an extended period of time (in this case 30 seconds), and therefore was used to adjust their vitals.

We can also see from Table 4.7 that the user's vitals were extremely close to the targets given to the Osti realtime application. If we compute the standardised cosine similarity and Euclidiean distance of these from the targets in the same way as in Section 4.2.1.1 then we obtain a cosine similarity of 0.878 and a Euclidian distance of 0.950. The cosine similarity and Euclidian distance of the Osti-generated playlist were 0.688 and 1.011 respectively, which shows that for this workout type, the real-time adjustments helped keep the user closer to their target vital values.

	Duration (mins)	**Average Heart Rate (bpm)**	**Calories Burned (kcal)**	**Distance Covered (m)**	**Average Speed (m/s)**	**Steps Taken**
Target	20	150	242.2	2700	2.2	2700
Playlist Results	21.1	149.9	229.7	2713.7	2.17	2826
Real-time Results	20.2	152.5	230.0	2702.2	2.20	2743

Table 4.7: Running vitals for Osti playlist and real-time recommendations



(a) Time vs Heart Rate

(b) Time vs Distance Travelled

(c) Time vs Speed

(d) Time vs Calories Burned

Figure 4.13: User Vitals for a Running workout using real-time Osti recommendations

4.2.2.2 Strength Training

The second workout type we considered was strength training, which is lower intensity that running. It also doesn't involve constant exertion, since users usually train in sets, and therefore we wanted to see how this affected the Osti recommendation algorithm.

Figure 4.14 shows a plot of a user's heart rate and calories burned over time. The red, yellow and green lines represent the same as for the running graphs. Figure 4.14b, which shows the calories burned over time, isn't very useful, since it suggests the calories were burned at a constant rate. This is still useful in the algorithm, since this can be used to predict how close the user will be to the target number of calories, however is not useful for visualisation and analysis. Therefore, we will focus on Figure 4.14a, since heart rate varies a lot more drastically over time.

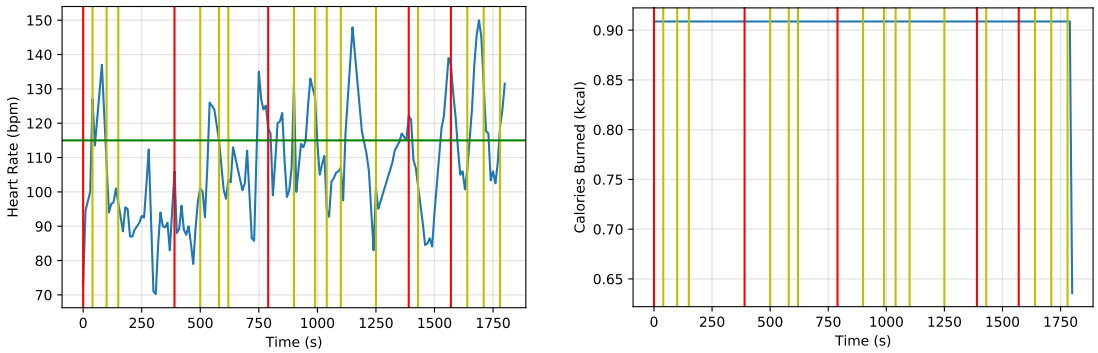
Figure 4.14a shows that only on 5 occasions was a track played the full way through. This equates to 75% of tracks eventually being skipped. This is much higher than in the running workout at only 50%, since the user is not showing constant exertion. As we can see, the heart rate fluctuates a lot more than for the running workout, since the user is completing the workout in sets, and therefore their heart rate will raise during this set, and then fall before the next one. This resulted in tracks changing frequently, since the user would stay above target for around 30 seconds (the length of time before a track skips if it is not ideal), causing the track to skip. The same situation would then repeat once they are resting in between sets. This resulted in track being skipped much more frequently in real time, resulting in 20 tracks being played, compared to only 9 in the playlisted workout. The user who completed this workout gave feedback that "The constant changing of tracks sometimes took me out of the zone and interrupted my workout", also suggesting that the tracks were changed too frequently.

Table 4.8 shows that the average heart rate during this workout was 107.3bpm, and the total number of calories burned was 164.2. As we can see, the real-time recommendations ended up with the number of calories burned much closer to the target, however it did result in a lower heart rate, which was 7.7bpm below target, vs only 2.6bpm below target during the playlist generated workout.

The fluctuations in heart rate discussed earlier due to the workout not having constant exertion are a likely cause for this, since the user explained that the frequent track skipping distracted them from the workout. This also means that it is less likely that the tracks were having the desired effect in raising and lowering the heart rate, and probably were more likely only influencing the maximum heart rates being reached. The standardised cosine similarity and Euclidian distance for the real-time workout were -0.618 and 3.045 respectively, which also suggest that the real-time recommendations did not keep the user as close to their target vitals as the playlist, which had values of 0.794 and 1.466.

	Duration (mins)	**Average Heart Rate (bpm)**	**Calories Burned (kcal)**
Target	30	115	170
Playlist Results	31.0	112.4	183.5
Real-time Results	30.1	107.3	164.2

Table 4.8: Strength Training vitals for Osti playlist and real-time recommendations



(a) Time vs Heart Rate

(b) Time vs Calories Burned

Figure 4.14: User Vitals for a Strength Training workout using real-time Osti recommendations

4.2.2.3 Overview

After comparing the real-time recommendation results for both a running and strength training workout, it is clear that the algorithm worked much better for the running workout, since it had a higher cosine similarity than all of the playlists, and a lower Euclidean distance. One predicted reason for this is because running workouts have a constant exertion. This allowed songs to be played for much longer, and actually have the desired effect on the user, as can be seen in Figure 4.13.

We conclude that real-time recommendations are not as suitable for workouts such as strength training, which are completed in sets, where vitals such as heart rate increase and decrease regularly independent of music. Further research could be done for methods which change tracks less frequently for workouts of non-constant exertion, or by not skipping tracks and instead just calculating the next song in real-time.

Chapter 5

Ethical Considerations

5.1 Human Participation

One key ethical consideration for the implementation of this project was the use of human participants, especially for the user testing of the application. Since this is a user-focused application, it's designed as an aid to help improve a user's workout performance by providing assistance through music. Therefore, for the evaluation, we tracked user vitals, such as heart rate, during workouts, to assess whether the suggested music helped to improve their performance. This was essential to evaluating the outcome of the project.

5.2 Data Protection

The project requires collecting personal data from users. The first kind of data collected is listening history; the songs a user has listened to and when they listened to them. This relies on two APIs: Last.FM and Spotify. The second kind of data collected is workout data, which is collected using the Google Fit API, as well as in real-time using the WatchOS application.

This data is also processed in order to find patterns which allow the recommendation engine to generate accurate song suggestions. This is needed for the application to fulfil its core functionality as a music recommendation service. Therefore, it must abide by the Data Protection Act, by being transparent with the user as to what data is being stored, why it is needed, and making sure to get their consent. This is done when the APIs are connected, since the users give permissions for Osti to access the APIs and collect appropriate data.

Sensitive data is also going to be collected, since health data is going to be used to assist in workout-specific recommendations. This therefore means that data needs to be stored securely, and users have the ability to have their data removed from the system upon request.

The application could also be classed as tracking users, since it will build a record of their music listening history, and will have access to workout data, so it can be seen what activities a user was doing at a particular time.

The music recommendation engine uses both use a combination of user history, as well as publicly available data sets, which allows it to generate recommendations which are more appropriate for a given workout. The publicly available data sets, such as the Spotify Million Playlist Dataset [29], is used for collaborative filtering, and will not be combined with user data in order to find users in these data sets.

Chapter 6

Discussion, Future Work and Conclusion

6.1 Discussion

As discussed in Section 4.2.1, Osti is successful in creating playlists which help a user to achieve their target vitals over the course of a workout. Compared to Spotify's Sountrack Your Workout playlists and generic workout playlists, Osti did a better job at helping a user reach their desired vitals by the end of a workout, and not under- or over-shoot them. User-curated workout playlists were the only playlist type which had similar results to Osti, since they've been curated by the user for the exact workout type, however in higher-intensity workouts Osti even outperformed these. This is because it can use collected data to detect the effect of tracks on workout performance.

Furthermore, we also reported how the real-time application of Osti's algorithm, as tested in Section 4.2.2, helped constant exertion workouts, such as running, to keep user vitals steady and on target. However, we also learned that in it's current state, the real-time algorithm is not as useful for workouts such as strength training, where user vitals fluctuate more drastically and repetitively over time, due to the workout generally happening in sets, where vitals peak and drop repeatedly. Further investigation is needed into techniques which could help to adapt Osti playlists to real-time for these types of workouts.

We also showed in Section 4.1 that Osti successfully managed to generate recommendations which had similar musical features to those which had been observed in each workout context. This is done using real data obtained from real test users, showing that the algorithms used by Osti are applicable in the real world, with data from a variety of sources. These visualisation results also showed that a hybrid approach of collaborative filtering and content-based recommendations can produce accurate recommendations that are later used in workout contexts.

Osti was also used to evaluate it's methods on non-workout contexts, such as studying, evaluated under the workout type 'mindfulness'. The musical features of the recommendations produced for this workout type aligned with what was expected, based on intensity of workout. This shows that the general approach of recommending tracks using user vitals works in contexts which have a lower intensity, and therefore the algorithm could be extended to contexts outside of workouts. For the workout-based collaborative filtering to still be usable, the current user context would still need to be obtained. Some other works, such as the system designed at the National University of Singapore [14], has had success with activity detection based on user vitals, and therefore Osti could be extended to incorporate this to allow for recommendations for a wider range of contexts.

6.2 Future Work

There are many different ways that Osti could be extended in the future to enable more advanced recommendations. These include being able to customise workout targets over time (Section 6.2.1), more research into real-time recommendations for workouts such as strength training (Section

6.2.2) integrating a temporal context into recommendations (Section 6.2.3), and understanding the subject of a track (Section 6.2.4).

6.2.1 Customisable Workout Target Vectors

Osti currently uses a set of up to five target vitals per user-workout pair to track workout performance: calories burned, average heart rate, distance, average speed and steps taken. The workout duration can also be set. However, these vitals are all constant values over the duration of the workout, which may not reflect how a user wants to set their targets.

Therefore, a useful extension would be to adapt the targets so that they do not need to be constant over time, and could be edited by the user for different workout styles. For example, a user may want to do a run where they sprint for 2 minutes, then run for 2 minutes, then walk for 2 minutes, and would like their targets and the tracks played to reflect this. Osti was built with extensibility in mind, and therefore already treats user workout targets as an array, where each value is the target over 10 seconds of the workout, as described in Section 3.4.1. This therefore would make it simple to extend to customisable workout target vectors.

6.2.2 Further Research into Real-Time Recommendations for Non-Constant Exertion Workouts

As mentioned in Section 4.2.2, the real-time Osti recommendation application works well for workouts where exertion is more constant, such as running. However, for workouts such as strength training, where users' heart rate and other vitals fluctuate due to completing exercise in sets, Osti often skips songs too frequently.

Further research could be done into methods which would allow for a more useful recommendation algorithm for this workout type. One option would be not not skip songs, and just calculate the next song based on current workout trends, and therefore the skipping of multiple songs in quick succession would not be a problem. Alternatively, the parameters determining how long a user has to be out of target range before a track is skipped could be learned by the system depending on workout type.

6.2.3 Temporal Recommendations

Currently, Osti takes in as much listening history as possible from a user's Last.FM account, in order to have the maximum amount of data to work with. However, this means that if this listening history is very extensive, potentially going back multiple years, then more recently discovered songs may be outweighed by older songs which have been played more.

Therefore, introducing a temporal aspect to the music recommendation engine would allow for a user's current music taste to be isolated and used in recommendations, since music taste can evolve over time [34]. It would also allow for playlists to be created based on listening history at a specific point in time, opening up the possibility of 'throwback' playlists. Success has been shown by Dias and Fonseca in using temporal context in collaborative filtering algorithms [39], and also neural methods have been researched by Lin et al. that have been shown to improve short-term recommendations (akin to the real-time implementation of Osti) using RNNs [40].

6.2.4 Lyric Comprehension & Music Tagging

Osti uses many features of tracks, such as tempo, artist and learned features like danceability and acousticness when recommending tracks. It also uses popularity metrics, as well as collaborative filtering to take advantage of other user's patterns. Another interesting way of categorising tracks would be to understand the subject of the track lyrics.

There are two potential methods of doing this. The first would be using some Natural Language Processing on the lyrics of a track to understand what the track is about. The other would be to use an API which gives user-annotated tags for the genre and theme of tracks, such as Last.FM. The advantage of using NLP would be that it could be applied to any track for which lyrics can be

retrieved, however it would then depend on how accurate the NLP model is at tagging the track. User-annotated tags would be more accurate, however would require a massive data source which is kept up to date, since Osti is designed to work with all tracks available on streaming services.

6.3 Conclusion

We've created an end-to-end system, Osti, which takes real user workout and listening data, collected automatically and in real-time, and uses this to compute recommendations for each workout type a user has completed before. From these recommendations, it generates both playlists of ideal tracks based on workout history, as well as adapts these ideal tracks in real-time based on current user vitals.

Using a hybrid approach of collaborative filtering and content-based recommendations, Osti avoids the cold start problem for initial users in the system, and can recommend full workout playlists from as little as one track of prior knowledge in a workout context. These recommendations can also be visualised by users through a web application, as well as manually adapted if a user disagrees with them.

The use of historical workout and listening data allows a workout profile to be built, and from this the effect of tracks on a user's workout performance can be learned. Combining these learnings with musical features, and using the profile of a users workout, along with their target vital values, a playlist of ideal songs can be curated to keep a user close to these values.

This algorithm was also adapted to work in real-time, in the form of an Apple Watch application. Using user vitals, such as heart rate, collected from the watch, the application calculates the most ideal track on device, and then uses Spotify's API to control the music playing in sync to the workout, by skipping tracks and reshuffling the queue to help the user close in on their targets.

Osti was evaluated using a combination of data visualisation and user testing. We showed that the features of the tracks being recommended aligned with those observed in the same workout context, and also followed expected patterns based on workout intensity. Through user testing, we showed that Osti-generated playlists outperformed other workout playlists for the workout types we tested, helping users keep close to their target vital values. They even performed equally as good as of slightly better than user-curated playlists. Real-time recommendations further improved how close user vitals were to target for constant-exertion workouts such as running.

Appendix A

Database Schema

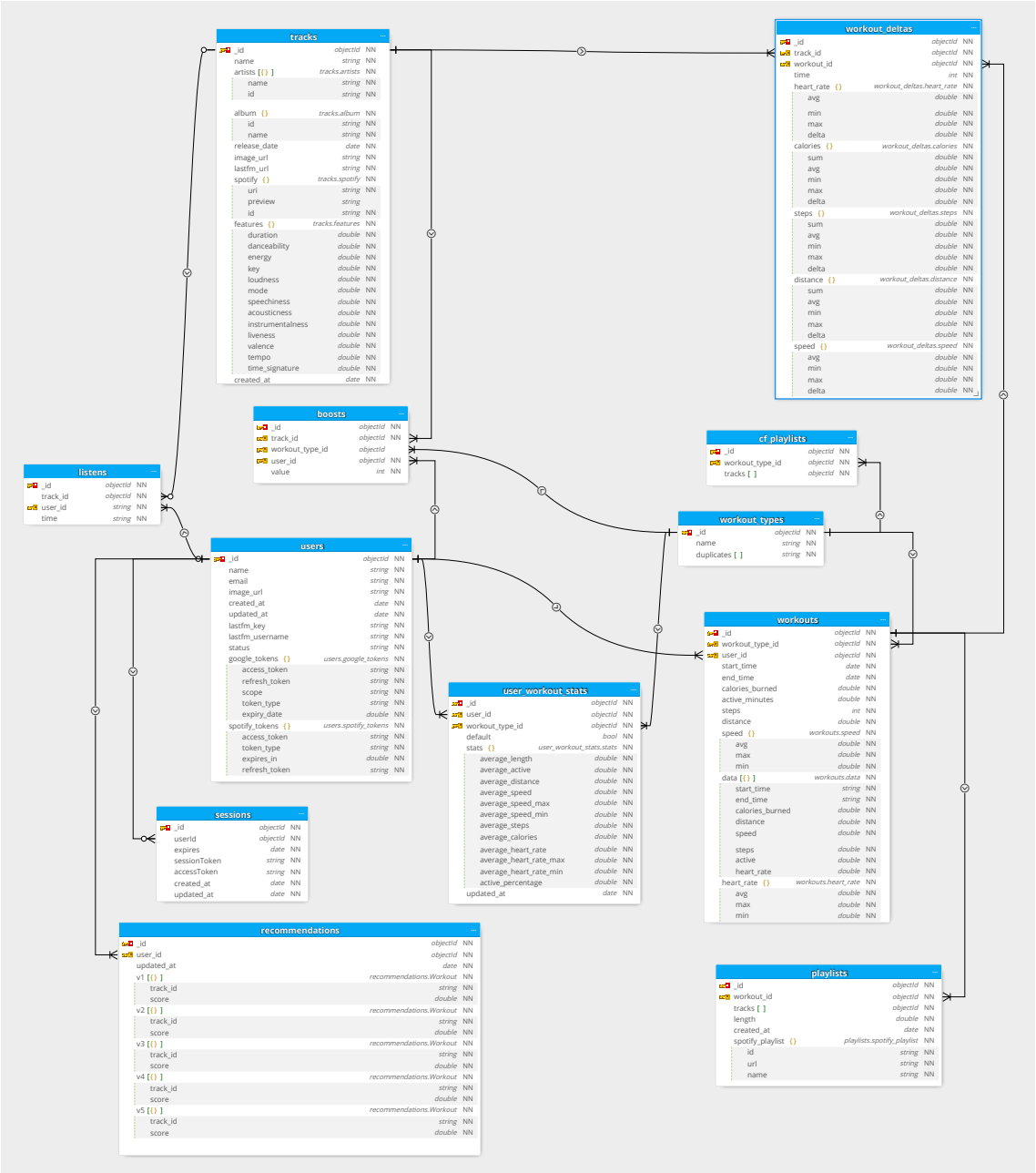


Figure A.1: Database Schema

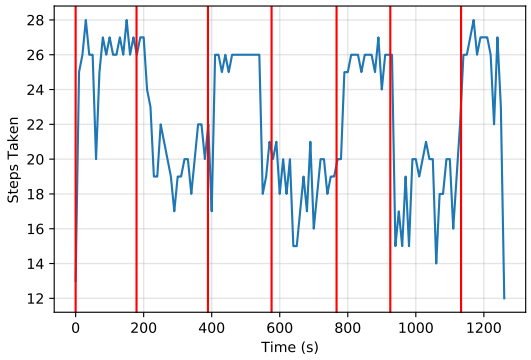
Appendix B

User Testing Results: Playlist Generation

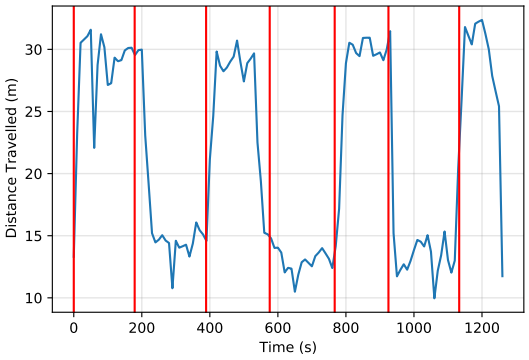
This appendix shows the detailed results of the user testing for playlist generation from 5 different playlist sources described in Section 4.2.1. It displays the plots of user vitals over the duration of a workout, along with the point at which each track starts, marked by a red line. It also includes a table of which tracks were played, and whether they've been heard by a user before in the context and in general. The graphs for calories burned have been excluded for walking and running workouts, since this vital had very small fluctuations, and therefore the graph did not show anything useful.

B.1 Osti Playlist

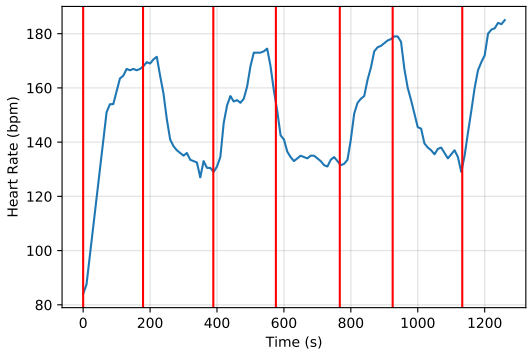
B.1.1 Running



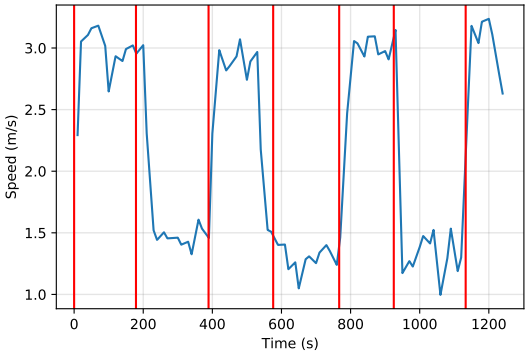
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



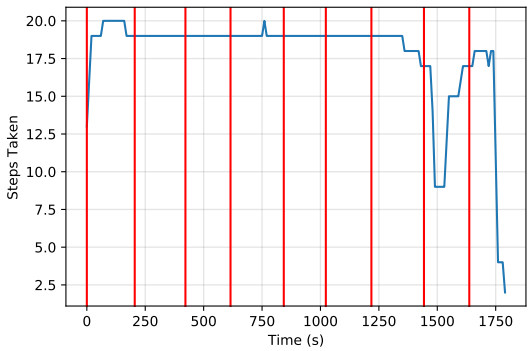
(d) Time vs Speed

Figure B.1: User Vitals for a Running Workout to Osti Playlist

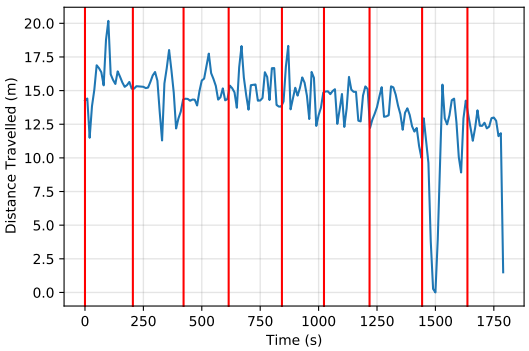
Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Dancing With Our Hands Tied	Taylor Swift	160.024	Yes	Yes
2	So Am I	Ava Max, NCT 127	130.05	No	Yes
3	Smooth Criminal (Glee Cast Version)	Glee Cast, 2CELLOS	135.043	Yes	Yes
4	Black And White	Niall Horan	147.589	No	No
5	2U - R3HAB Remix	David Guetta	144.978	Yes	Yes
6	Call It What You Want	Taylor Swift	163.954	Yes	Yes
7	Sweet Melody	Little Mix	119.965	No	Yes

Table B.1: Tracks listened to for a Running Workout to Osti Playlist

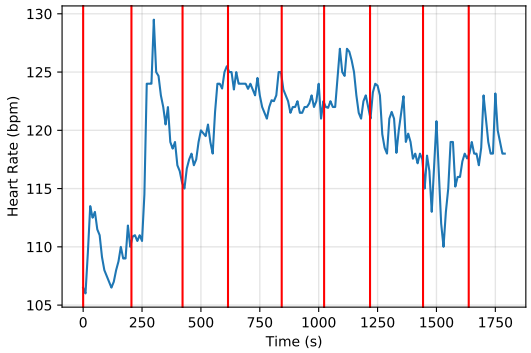
B.1.2 Walking



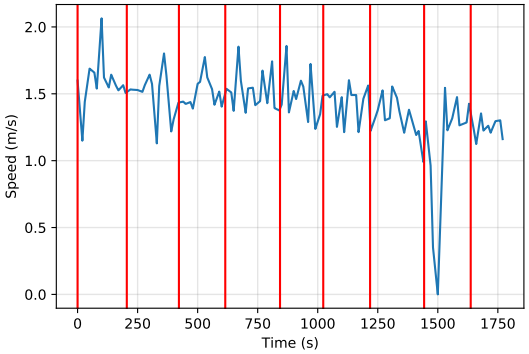
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



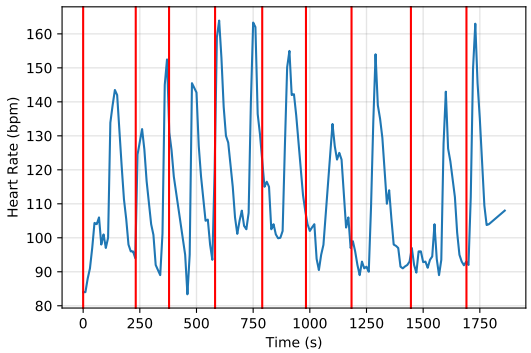
(d) Time vs Speed

Figure B.2: User Vitals for a Walking Workout to Osti Playlist

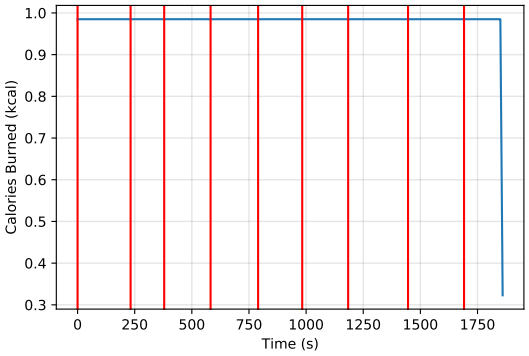
Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Wasabi	Little Mix	114.001	Yes	Yes
2	One Foot in Front of the Other	Griff	112.043	Yes	Yes
3	GUY.exe	Superfruit	110.011	Yes	Yes
4	MORE & MORE - English Version	TWICE	106.998	Yes	Yes
5	You	Regard	106.064	Yes	Yes
6	Confetti (feat. Saweetie)	Little Mix	107.088	Yes	Yes
7	Ciao Adios	Anne-Marie	106.083	No	Yes
8	Won't Go Home Without You	Maroon 5	110.022	No	No
9	Love Me Again	RAYE	110.063	No	Yes

Table B.2: Tracks listened to for a Walking Workout to Osti Playlist

B.1.3 Strength Training



(a) Time vs Heart Rate



(b) Time vs Calories Burned

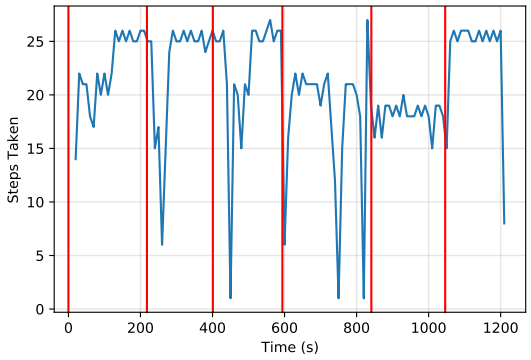
Figure B.3: User Vitals for a Strength Training Workout to Osti Playlist

Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Whole Lotta Woman	Kelly Clarkson	119.982	Yes	Yes
2	Electricity	Silk City, Dua Lipa	118.159	Yes	Yes
3	No Excuses	Meghan Trainor	115.022	Yes	Yes
4	Starstruck	Years & Years	113.852	Yes	Yes
5	Number 1	Tinchy Stryder	114.912	Yes	Yes
6	Dynamite	BTS	114.044	Yes	Yes
7	Say So	Doja Cat, Nicki Minaj	111.004	Yes	Yes
8	Forever & Always (Piano Version) (Taylor's Version)	Taylor Swift	118.753	No	Yes
9	I Don't Want It At All	Kim Petras	110.018	Yes	Yes

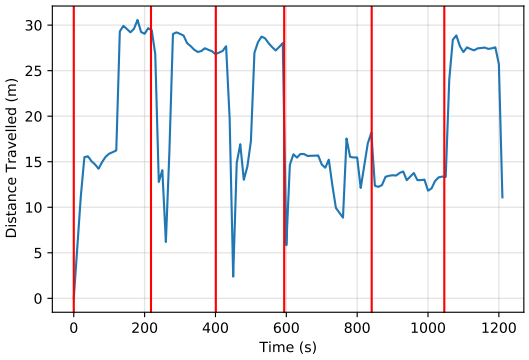
Table B.3: Tracks listened to for a Strength Training Workout to Osti Playlist

B.2 Spotify's Soundtrack Your Workout Playlist

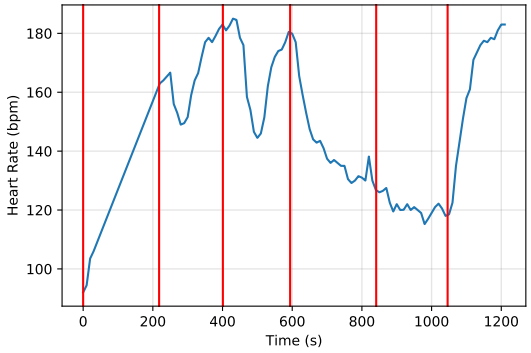
B.2.1 Running



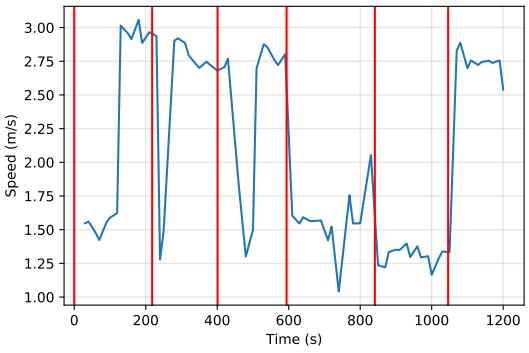
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



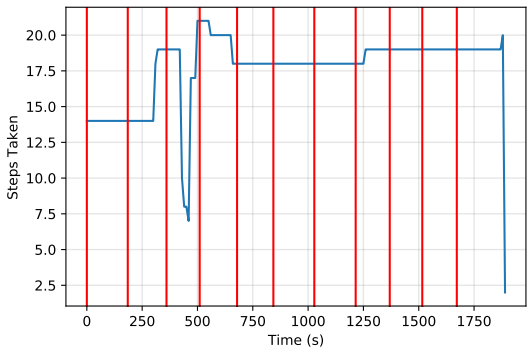
(d) Time vs Speed

Figure B.4: User Vitals for a Running Workout to Spotify's Soundtrack Your Workout Playlist

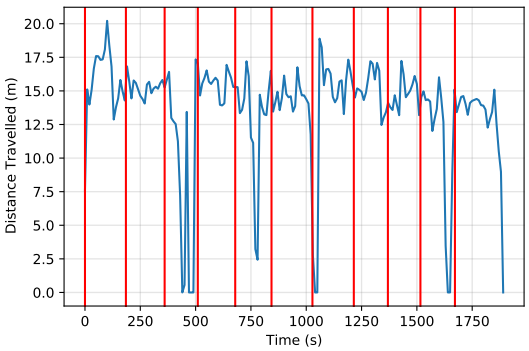
Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	LUCID	Rina Sawayama	125.98	No	Yes
2	Rain On Me	Lady Gaga, Ariana Grande	123.056	Yes	Yes
3	Motivation	Normani	170.918	No	Yes
4	Shout Out to My Ex	Little Mix	126.014	No	Yes
5	Safe With Me	Gryffin, Audrey Mika	169.902	No	No
6	Feels In My Body	Icona Pop	122.047	No	Yes

Table B.4: Tracks listened to for a Running Workout to Spotify's Soundtrack Your Workout Playlist

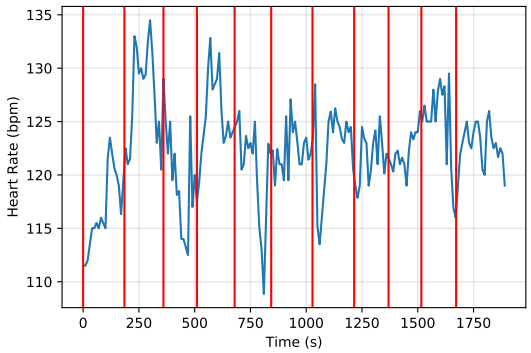
B.2.2 Walking



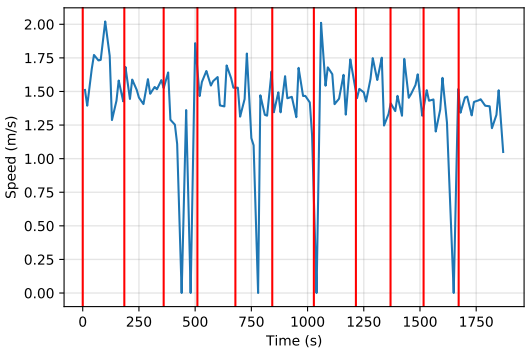
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



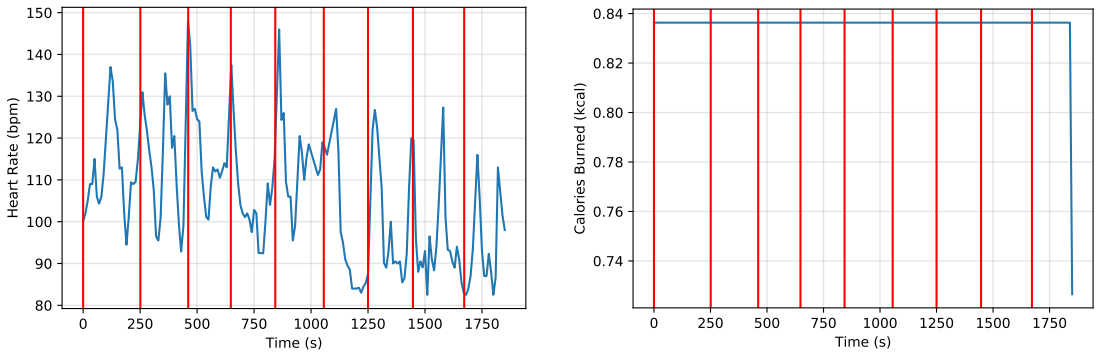
(d) Time vs Speed

Figure B.5: User Vitals for a Walking Workout to Spotify's Soundtrack Your Workout Playlist

Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Sweet Melody	Little Mix	119.965	Yes	Yes
2	Save Your Tears (Remix)	The Weeknd, Ariana Grande	118.091	Yes	Yes
3	Look What You've Done	Zara Larsson	117.915	Yes	Yes
4	Leave Before You Love Me	Marshmello, Jonas Brothers	119.976	No	No
5	My Head & My Heart	Ava Max	116.001	No	No
6	Forever	FLETCHER	115.937	No	No
7	Sad Songs	Route 94, L Devine	123.986	No	No
8	Malibu	Kim Petras	120.018	Yes	Yes
9	Feels In My Body	Icona Pop	122.047	No	Yes
10	Lifestyle	Jason Derulo, Adam Levine	123.055	No	No
11	Summer Feelings	Lennon Stella, Charlie Puth	115.982	No	No

Table B.5: Tracks listened to for a Walking Workout to Spotify's Soundtrack Your Workout Playlist

B.2.3 Strength Training



(a) Time vs Heart Rate

(b) Time vs Calories Burned

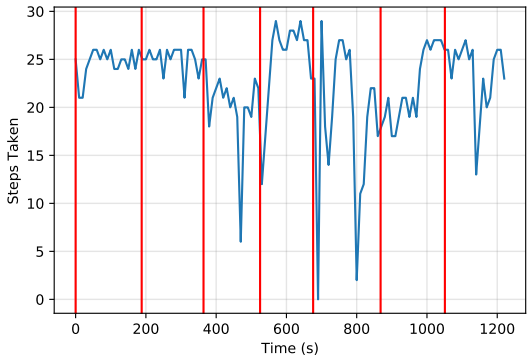
Figure B.6: User Vitals for a Strength Training Workout to Spotify's Soundtrack Your Workout Playlist [23]

Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Rain On Me	Lady Gaga, Ariana Grande	123.056	Yes	Yes
2	Shout Out to My Ex	Little Mix	126.014	Yes	Yes
3	Fallin' (Adrenaline)	Why Don't We	133.963	No	No
4	Malibu	Kim Petras	120.018	No	Yes
5	Birthday	Anne-Marie	151.995	Yes	No
6	Love Myself	Hailee Steinfeld	122.924	Yes	Yes
7	Ugly Heart	G.R.L.	124.993	No	No
8	She Looks So Perfect	5 Seconds of Summer	160.025	No	Yes
9	Last Friday Night (T.G.I.F.)	Katy Perry	126.023	No	Yes

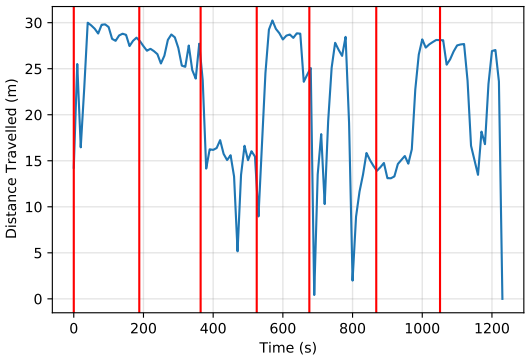
Table B.6: Tracks listened to for a Strength Training Workout to Spotify's Soundtrack Your Workout Playlist [23]

B.3 User Hand-Curated Playlist

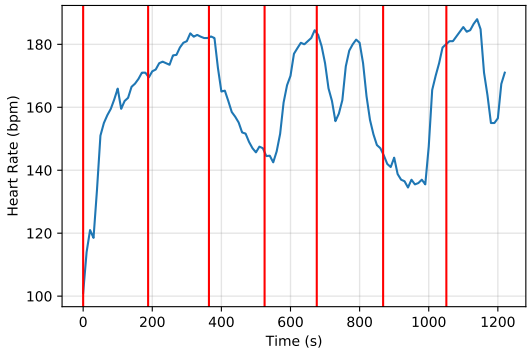
B.3.1 Running



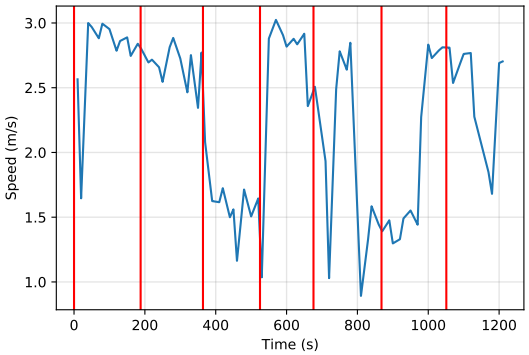
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



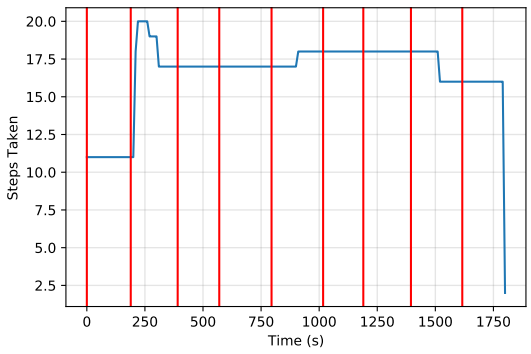
(d) Time vs Speed

Figure B.7: User Vitals for a Running Workout to a User Hand-Curated Playlist

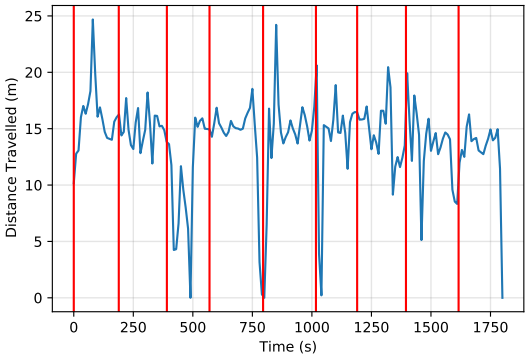
Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	2U (feat. Justin Bieber)	David Guetta	144.937	Yes	Yes
2	Rain On Me	Lady Gaga, Ariana Grande	123.056	Yes	Yes
3	UK Hun? (United Kingdolls Version)	The Cast of RuPaul's Drag Race UK, Season 2	134.038	No	Yes
4	Up	Cardi B	166	No	Yes
5	Happiness	Little Mix	125.932	Yes	Yes
6	Decline	RAYE	115.924	No	Yes
7	WOW	Zara Larsson	77.45	Yes	Yes

Table B.7: Tracks listened to for a Running Workout to a User Hand-Curated Playlist

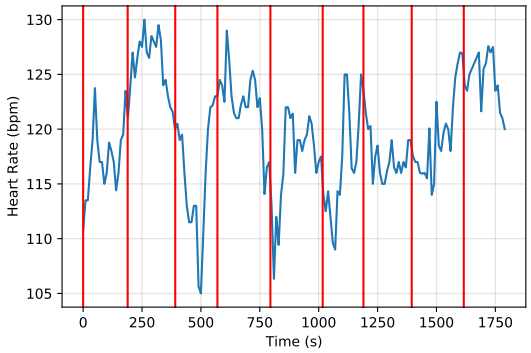
B.3.2 Walking



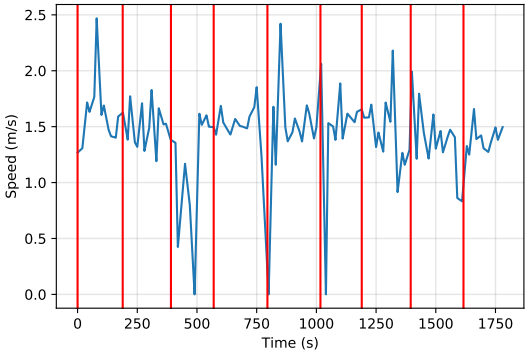
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



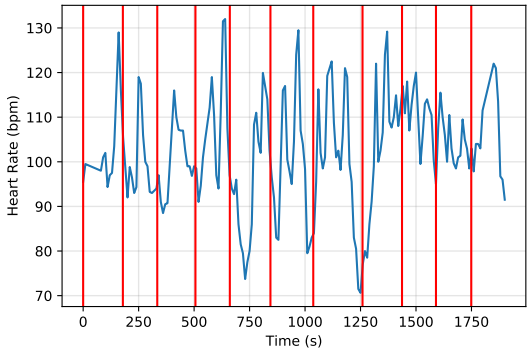
(d) Time vs Speed

Figure B.8: User Vitals for a Walking Workout to a User Hand-Curated Playlist

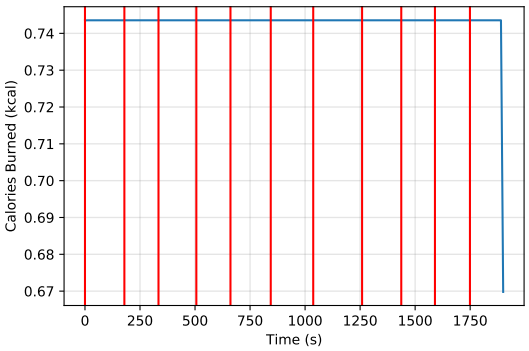
Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Save Your Tears (Remix)	The Weeknd, Ariana Grande	118.091	Yes	Yes
2	I Really Like You	Carly Rae Jepsen	122.121	Yes	Yes
3	The Motto	Drake	201.8	Yes	Yes
4	Bad 4 Us	Superfruit	110.012	Yes	Yes
5	Question	Alex Aiono	100.029	Yes	Yes
6	Into It	Camila Cabello	108.77	Yes	Yes
7	Tongue	MNEK	106.456	Yes	Yes
8	Never Really Over	Katy Perry	99.991	Yes	Yes
9	Look At Her Now	Selena Gomez	77.342	Yes	Yes

Table B.8: Tracks listened to for a Walking Workout to a Generic Workout Playlist ('Walking Music' Playlist)

B.3.3 Strength Training



(a) Time vs Heart Rate



(b) Time vs Calories Burned

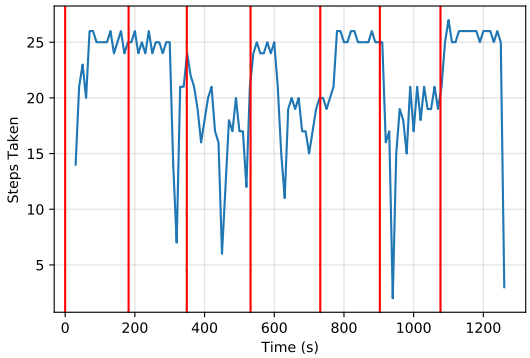
Figure B.9: User Vitals for a Strength Training Workout to a User Hand-Curated Playlist

Position	**Track Name**	**Track Artist**	**BPM**	**Known by User in Context**	**Known by User**
1	Somebody	Justin Bieber	75.991	Yes	Yes
2	Heartbreak Anthem	Galantis, David Guetta, Little Mix	124.111	Yes	Yes
3	Right Here - Alok Remix	Zara Larsson	121.973	Yes	Yes
4	good 4 u	Olivia Rodrigo	168.56	Yes	Yes
5	Love Me Land	Zara Larsson	216.334	Yes	Yes
6	Don't Play	Anne-Marie	128.031	No	Yes
7	Gonna Get This	Hannah Montana	100.02	No	Yes
8	All the Time	Zara Larsson	101.976	Yes	Yes
9	Valentino	Years & Years	111.053	No	Yes
10	MONOPOLY	Ariana Grande, Victoria Monét	143.96	No	Yes
11	No Time For Tears	Nathan Dawe, Little Mix	125.009	Yes	Yes

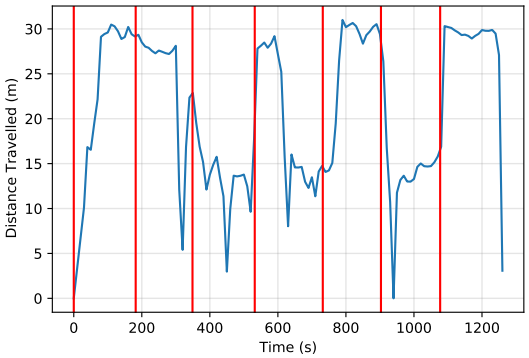
Table B.9: Tracks listened to for a Strength Training Workout to a User Hand-Curated Playlist

B.4 Generic Workout Playlist

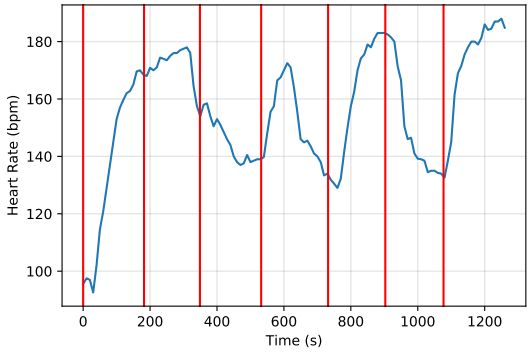
B.4.1 Running: 'Fun Run' Playlist



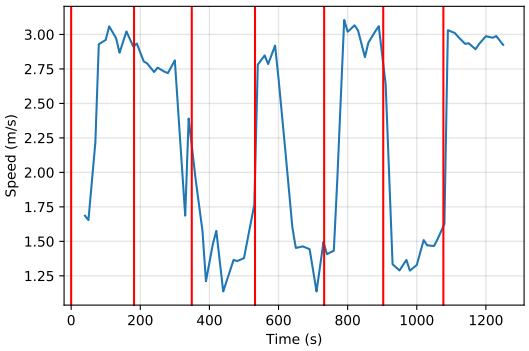
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



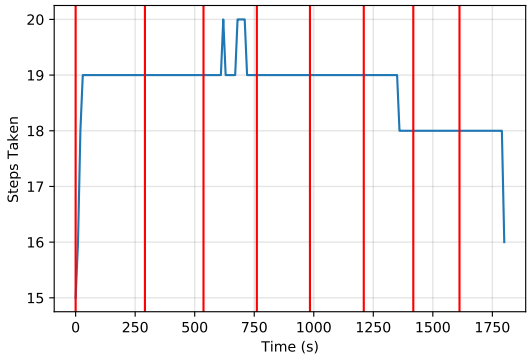
(d) Time vs Speed

Figure B.10: User Vitals for a Running Workout to a Generic Workout Playlist ('Fun Run' Playlist)

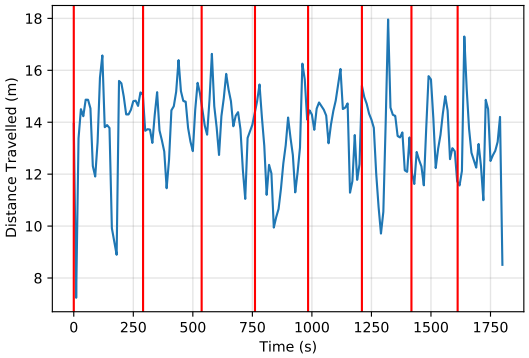
Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Black and White	Niall Horan	147.589	No	No
2	Stupid Love	Lady Gaga	117.987	Yes	Yes
3	Don't Start Now	Dua Lipa	123.95	No	Yes
4	Sucker	Jonas Brothers	137.958	No	Yes
5	Youngblood	5 Seconds of Summer	120.274	No	Yes
6	Somebody	Dagny	117.981	No	No
7	Don't Call Me Up	Mabel	98.994	No	Yes

Table B.10: Tracks listened to for a Running Workout to a Generic Workout Playlist ('Fun Run' Playlist)

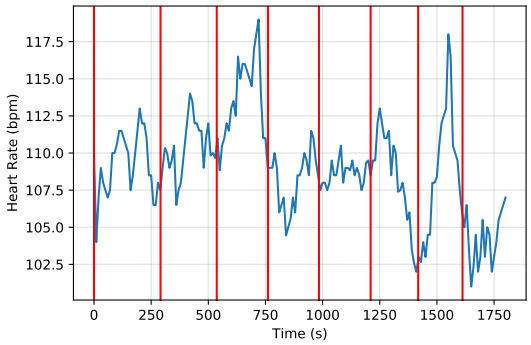
B.4.2 Walking: 'Walking Music' Playlist



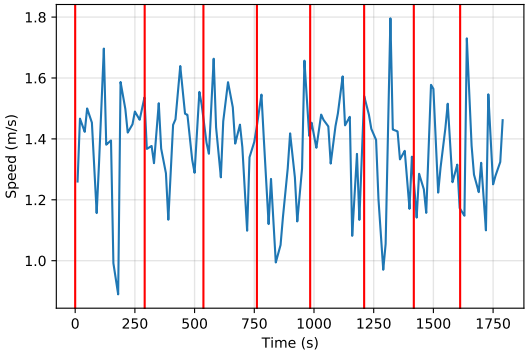
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



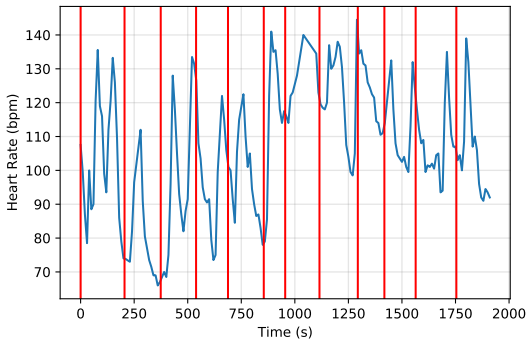
(d) Time vs Speed

Figure B.11: User Vitals for a Walking Workout to a Generic Workout Playlist ('Walking Music' Playlist)

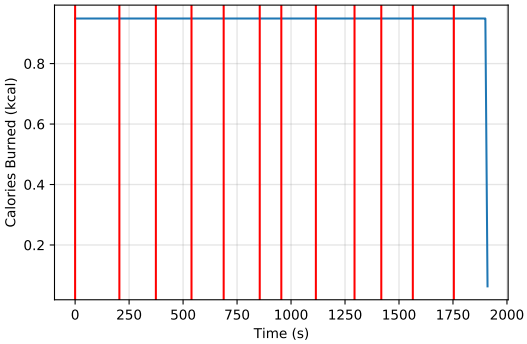
Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	The Winner Takes It All	ABBA	126.15	No	No
2	I've Done Everything for You	Rick Springfield	161.512	No	No
3	Don't Stop Believin'	Journey	118.852	No	No
4	Only Wanna Be with You	Hootie & The Blowfish	103.272	No	No
5	Animals	Maroon 5	189.868	No	Yes
6	Won't Go Home Without You	Maroon 5	110.022	No	No
7	Rhythm of Love / Can't Help Falling in Love - EP Version	Straight No Chaser	85.483	No	No

Table B.11: Tracks listened to for a Walking Workout to a Generic Workout Playlist ('Walking Music' Playlist)

B.4.3 Strength Training: 'Workout' Playlist



(a) Time vs Heart Rate



(b) Time vs Calories Burned

Figure B.12: User Vitals for a Strength Training Workout to a Generic Workout Playlist ('Workout' Playlist)

Position	Track Name	Track Artist	BPM	Known by User in Context	Known by User
1	Met Him Last Night	Demi Lovato, Ariana Grande	144.978	Yes	Yes
2	LazyBaby	Dove Cameron	113.019	No	No
3	good 4 u	Olivia Rodrigo	168.56	Yes	Yes
4	Overdrive	Conan Gray	104.959	No	Yes
5	BOY	Madison Beer	121.122	No	Yes
6	Not a Pop Song	Little Mix	139.115	Yes	Yes
7	test drive	Ariana Grande	115.036	No	Yes
8	Hold On	Justin Bieber	140.002	No	Yes
9	The Man	Taylor Swift	110.048	No	Yes
10	Obsessed	Addison Rae	110.016	No	No
11	Sacrifice	Bebe Rexha	119.992	No	No
12	Happiness	Little Mix	125.932	Yes	Yes

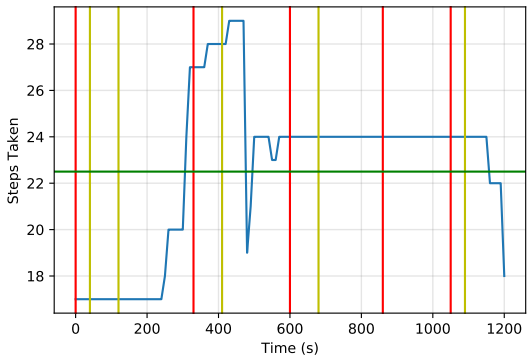
Table B.12: Tracks listened to for a Strength Training Workout to a Generic Workout Playlist ('Workout' Playlist)

Appendix C

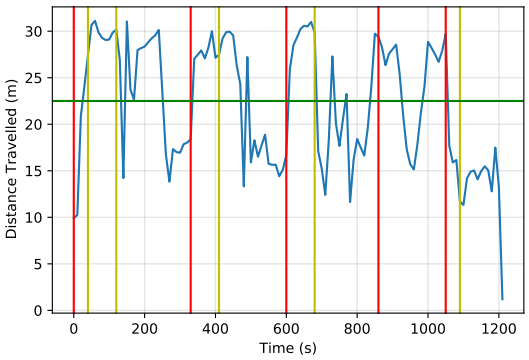
User Testing Results: Real-Time Recommendations

This appendix shows the detailed results of the user testing for real-time track recommendations described in Section 4.2.2. It displays the plots of user vitals over the duration of a workout, along with the point at which each track starts, where red lines represent a track which has started naturally (either the initial track, or has started following the finishing of the previous track), and yellow lines represent a track which was started due to the previous one being skipped. It also includes a table of which tracks were played, at which time, whether they were skipped or not, and whether they've been heard by a user before in the context and in general.

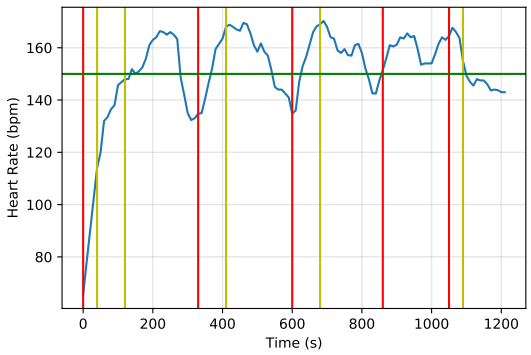
C.1 Running



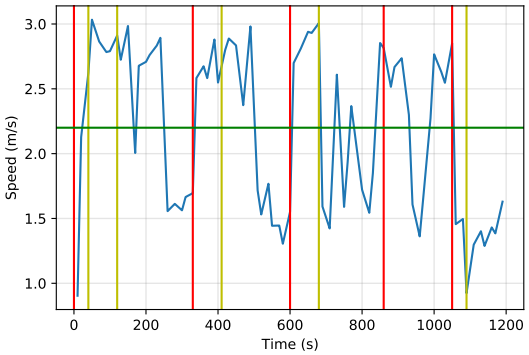
(a) Time vs Steps Taken



(b) Time vs Distance Travelled



(c) Time vs Heart Rate



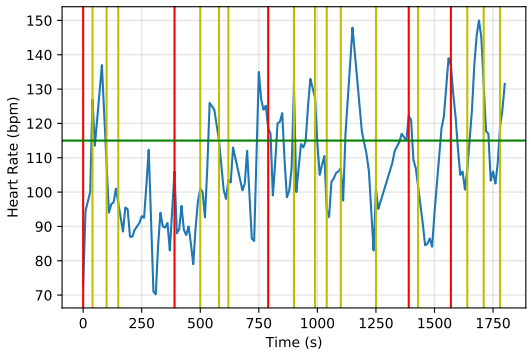
(d) Time vs Speed

Figure C.1: User Vitals for a Running Workout using the Osti real-time application

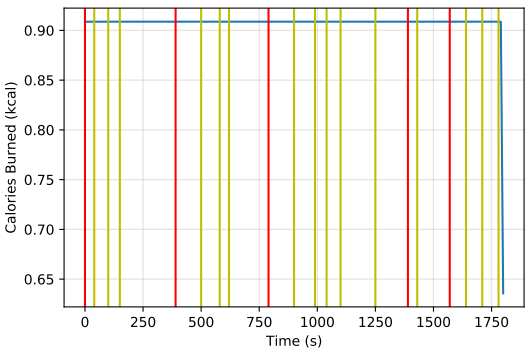
Start Time (secs)	Track Name	Track Artist	Tempo (bpm)	Skipped	Known by User in Context	Known by User
0	Dancing With Our Hands Tied	Taylor Swift	160.024	Yes	Yes	Yes
40	2U - R3HAB Remix	David Guetta, Justin Bieber	144.978	Yes	Yes	Yes
120	Smooth Criminal (Glee Cast Version)	Glee Cast, 2CELLOS	135.043	No	Yes	Yes
330	Don't Play	Anne-Marie	128.031	Yes	No	Yes
410	Call It What You Want	Taylor Swift	163.954	No	Yes	Yes
600	Better Than Revenge	Taylor Swift	145.821	Yes	No	Yes
680	Everyday	Ariana Grande	131.004	No	Yes	Yes
860	Happiness	Little Mix	125.932	No	Yes	Yes
1050	Don't Start Now	Dua Lipa	123.95	Yes	No	Yes
1090	Doctor You	DNCE	123.022	No	Yes	Yes

Table C.1: Tracks listened to for a Running Workout using the Osti real-time application

C.2 Strength Training



(a) Time vs Heart Rate



(b) Time vs Calories Burned

Figure C.2: User Vitals for a Strength Training Workout using the Osti real-time application

Start Time (secs)	Track Name	Track Artist	Tempo (bpm)	Skipped	Known by User in Context	Known by User
0	Whole Lotta Woman	Kelly Clarkson	119.982	Yes	Yes	Yes
40	Not a Pop Song	Little Mix	139.115	Yes	Yes	Yes
100	Everyday	Ariana Grande	131.004	Yes	Yes	Yes
150	Fearless (Taylor's Version)	Taylor Swift	100.118	No	Yes	Yes
390	1999	Charli XCX	124.016	Yes	Yes	Yes
500	Poster Girl	Zara Larsson	119.907	Yes	Yes	Yes
580	The Story Of Us	Taylor Swift	139.898	Yes	No	Yes
620	Hey Stephen (Taylor's Version)	Taylor Swift	115.99	No	Yes	Yes
790	Right Here	Zara Larsson	120.043	Yes	Yes	Yes
900	The Way You Felt	Alec Benjamin	131.918	Yes	Yes	Yes
990	Overdrive	Conan Gray	104.959	Yes	No	Yes
1040	Black Hole	Griff	124.069	Yes	Yes	Yes
1160	Morning	Zara Larsson	106.309	Yes	Yes	Yes
1250	New Me	Ella Eyre	115.444	No	Yes	Yes
1390	Dynamite	BTS	114.044	Yes	Yes	Yes
1430	LazyBaby	Dove Cameron	113.019	No	No	No
1570	Rain On Me	Lady Gaga, Ariana Grande	123.056	Yes	Yes	Yes
1640	Don't Let Me Be Yours	Zara Larsson	112.067	Yes	No	Yes
1710	I Need Love	Zara Larsson	135.948	Yes	No	Yes
1780	34+35	Ariana Grande	109.978	No	Yes	Yes

Table C.2: Tracks listened to for a Strength Training Workout using the Osti real-time application

Bibliography

[1] T Ingham. In a&r, 'gut vs. data' isn't a binary choice., May 2018. URL https://www.musicbusinessworldwide.com/in-ar-gut-vs-data-isnt-actually-a-binary-choice/. [Accessed 20th November 2020].

[2] C Karageorghis, P Terry, A Lane, D Bishop, and D Priest. The bases expert statement on use of music in exercise. *Journal of Sports Sciences*, 30(9):953–956, 2012. doi: 10.1080/02640414.2012.676665. URL https://doi.org/10.1080/02640414.2012.676665. PMID: 22512537.

[3] VM Nethery. Competition between internal and external sources of information during exercise: influence on rpe and the impact of the exercise load. *The Journal of sports medicine and physical fitness*, 42(2):172—178, June 2002. ISSN 0022-4707. URL http://europepmc.org/abstract/MED/12032412.

[4] D Priest and C Karageorghis. A qualitative investigation into the characteristics and effects of music accompanying exercise. *European Physical Education Review*, 14(3):347–366, 2008. doi: 10.1177/1356336X08095670. URL https://doi.org/10.1177/1356336X08095670.

[5] C Karageorghis, D Mouzourides, D Priest, T A Sasso, D J Morrish, and C J Walley. Psychophysical and ergogenic effects of synchronous music during treadmill walking. *Journal of sport & exercise psychology*, 31 1:18–36, 2009.

[6] D Wang, S Deng, and G Xu. Sequence-based context-aware music recommendation. *Information Retrieval Journal*, 21(2):230–252, Jun 2018. ISSN 1573-7659. doi: 10.1007/s10791-017-9317-7. URL https://doi.org/10.1007/s10791-017-9317-7.

[7] J Leskovec, A Rajaraman, and J D Ullman. *Mining of Massive Datasets*. Cambridge University Press, USA, 2nd edition, 2014. ISBN 1107077230.

[8] J H Su and T W Chiu. An item-based music recommender system using music content similarity. In N T Nguyen, B Trawiński, H Fujita, and T P Hong, editors, *Intelligent Information and Database Systems*, pages 179–190, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[9] Content-based filtering advantages & disadvantages, Nov 2018. URL https://developers.google.com/machine-learning/recommendation/content-based/summary. [Accessed 22nd December 2020].

[10] Collaborative filtering advantages & disadvantages, Feb 2020. URL https://developers.google.com/machine-learning/recommendation/collaborative/summary. [Accessed 22nd December 2020].

[11] L Zahrotun. Comparison jaccard similarity, cosine similarity and combined both of the data clustering with shared nearest neighbor method. *Computer Engineering and Applications Journal*, 5:11–18, 01 2016. doi: 10.18495/comengapp.v5i1.160.

[12] S Glen. Jaccard index / similarity coefficient, Dec 2016. URL https://www.statisticshowto.com/jaccard-index/. [Accessed 29th December 2020].

[13] A Heidarian and M Dinneen. A hybrid geometric approach for measuring similarity level among documents and document clustering. pages 142–151, 03 2016. doi: 10.1109/BigDataService.2016.14.

[14] 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*, MM '12, page 99–108, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450310895. doi: 10.1145/2393347.2393368. URL https://doi.org/10.1145/2393347.2393368.

[15] Ò Celma. *Music Recommendation*, pages 43–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13287-2. doi: 10.1007/978-3-642-13287-2_3. URL https://doi.org/10.1007/978-3-642-13287-2_3.

[16] P Knees and M Schedl. A survey of music similarity and recommendation from music context data. *ACM Trans. Multimedia Comput. Commun. Appl.*, 10(1), December 2013. ISSN 1551-6857. doi: 10.1145/2542205.2542206. URL https://doi.org/10.1145/2542205.2542206.

[17] N Hariri, B Mobasher, and R Burke. Context-aware music recommendation based on latenttopic sequential patterns. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, page 131–138, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312707. doi: 10.1145/2365952.2365979. URL https://doi.org/10.1145/2365952.2365979.

[18] S Nirjo, R F Dickerson, Q Li, P Asare, J A Stankovic, D Hong, B Zhang, X Jiang, F Shen, and F Zhao. Musicalheart: A hearty way of listening to music. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, page 43–56, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450311694. doi: 10.1145/2426656.2426662. URL https://doi.org/10.1145/2426656.2426662.

[19] A Khushhal, S Nichols, W Evans, D O Gleadall-Siddall, R Page, A F O'Doherty, S Carroll, L Ingle, and G Abt. Validity and reliability of the apple watch for measuring heart rate during exercise. *Sports medicine international open*, 1(6):E206–E211, Oct 2017. ISSN 2367-1890. doi: 10.1055/s-0043-120195. URL https://pubmed.ncbi.nlm.nih.gov/30539109.

[20] D J Spajic. Spotify user statistics that will be music to your ears, Dec 2020. URL https://kommandotech.com/statistics/spotify-user-statistics/. [Accessed 10th November 2020].

[21] H. Tankovska. Share of respondents who own a smart watch/health-tracker in uk 2019, by generation, Sep 2020. URL https://www.statista.com/statistics/1044033/uk-smartwatch-health-trackert-ownership/. [Accessed 22nd October 2020].

[22] Smartwatch market: 2020-2027: Industry report, 2019. URL https://www.mordorintelligence.com/industry-reports/smartwatch-market. [Accessed 25th October 2020].

[23] Revamp your fitness routine with soundtrack your workout, Jul 2020. URL https://www.newsroom.spotify.com/2020-07-08/revamp-your-fitness-routine-with-soundtrack-your-workout/. [Accessed 20th October 2020].

[24] Counterpoint Media. Music streaming market share, Apr 2020. URL https://www.statista.com/statistics/653926/music-streaming-service-subscriber-share/. [Accessed 25th May 2021].

[25] Strategy Analytics. Market share of smartwatch unit shipments worldwide from the 2q'14 to 1q'20, by vendor, May 2020. URL https://www.statista.com/statistics/524830/global-smartwatch-vendors-market-share/. [Accessed 25th May 2021].

[26] Spotify for Developers. Search for an item. URL https://developer.spotify.com/console/get-search-item/. [Accessed 11th June 2021].

[27] Spotify. Web api reference, . URL https://developer.spotify.com/documentation/web-api/reference/#object-audiofeaturesobject. [Accessed 25th May 2021].

[28] B Brost, R Mehrotra, and T Jehan. The music streaming sessions dataset. In *Proceedings of the 2019 Web Conference*. ACM, 2019.

[29] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. Recsys challenge 2018: Automatic music playlist continuation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, page 527–528, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450359016. doi: 10.1145/3240323.3240342. URL https://doi.org/10.1145/3240323.3240342.

[30] H Zamani, M Schedl, P Lamere, and C Chen. An analysis of approaches taken in the acm recsys challenge 2018 for automatic music playlist continuation. *ACM Trans. Intell. Syst. Technol.*, 10(5), September 2019. ISSN 2157-6904. doi: 10.1145/3344257. URL https://doi.org/10.1145/3344257.

[31] K. Watanabe, Y. Ooishi, and M. Kashino. Heart rate responses induced by acoustic tempo and its interaction with basal heart rate. *Sci Rep*, 7:43856, 03 2017.

[32] Apple. Speedysloth: Creating a workout. URL https://developer.apple.com/documentation/healthkit/workouts_and_activity_rings/speedysloth_creating_a_workout. [Accessed 1st June 2021].

[33] Vercel. Next.js by vercel - the react framework. URL https://nextjs.org/. [Accessed 28th May 2021].

[34] Alex Hsieh, Owen Luo, and Tyler Pavay. Music attributes and its effect on popularity. Dec 2020. URL https://ahsieh53632.github.io/music-attributes-and-popularity/. [Accessed 5th June 2021].

[35] Xavier Sanchez, Samantha Moss, Craig Twist, and Costas Karageorghis. On the role of lyrics in the music–exercise performance relationship. *Psychology of Sport and Exercise*, 15:132–138, 01 2014. doi: 10.1016/j.psychsport.2013.10.007.

[36] Spotify. Fun run playlist, . URL https://open.spotify.com/playlist/37i9dQZF1DXadOVCgGhS7j?si=80a4b648e2cb459b. [Accessed 6th June 2021].

[37] Spotify. Walking music playlist, . URL https://open.spotify.com/playlist/3yiX3ROHK4vo82pR6BO8eW?si=e7650819b8db4ef2. [Accessed 6th June 2021].

[38] Spotify. Workout playlist, . URL https://open.spotify.com/playlist/37i9dQZF1DX70RN3TfWWJh?si=47f03fb771ec4065. [Accessed 6th June 2021].

[39] Ricardo Dias and Manuel J. Fonseca. Improving music recommendation in session-based collaborative filtering by using temporal context. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 783–788, 2013. doi: 10.1109/ICTAI.2013.120.

[40] Qika Lin, Yaoqiang Niu, Yifan Zhu, Hao Lu, Keith Zvikomborero Mushonga, and Zhendong Niu. Heterogeneous knowledge-based attentive neural networks for short-term music recommendations. *IEEE Access*, 6:58990–59000, 2018. doi: 10.1109/ACCESS.2018.2874959.