BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

ANDROID MUSIC PLAYER WITH THE SONG SELECTION BY A DEVICE CONTEXT

PŘEHRÁVAČ HUDBY PRO ANDROID S VÝBĚREM SKLADEB DLE KONTEXTU ZAŘÍZENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. GABRIELA CHMELAŘOVÁ

SUPERVISOR
VEDOUCÍ PRÁCE

RNDr. MAREK RYCHLÝ, Ph.D.

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Information Systems (DIFS) Academic year 2020/2021

Master's Thesis Specification

22236

Student:	**Chmelařová Gabriela, Bc.**
Programme:	Information Technology
Field of study:	Intelligent Devices
Title:	**Android Music Player with the Song Selection by a Device Context**
Category:	User Interfaces

Assignment:

1. Study multimedia application development and device context detection for the Android operating system. Explore existing approaches and applications that use device context to change their activity.
2. Explore the possibilities of machine learning and suggest a suitable way to use it when selecting a song according to the context of the device.
3. After consulting with the supervisor, design and implement a music player for the Android operating system that will enable user driven selection of songs and, after learning period, it will select songs depending on the device context using machine learning techniques.
4. Test the solution, evaluate and discuss the results. Publish the resulting software as open-source.

Recommended literature:

- Ľuboslav Lacko. Vývoj aplikací pro Android. Computer Press, Brno, 2015. ISBN 978-80-251-4347-6.
- Salma Elmalaki, Lucas Wanner a Mani Srivastava. CAreDroid: Adaptation Framework for Android Context-Aware Applications. In: Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom '15). ACM, New York, NY, USA, s. 386-399, 2015. Online: [http://dx.doi.org/10.1145/2789168.2790108]
- Rahul Ravindran; Riya Suchdev; Yash Tanna a Sridhar Swamy. Context aware and pattern oriented machine learning framework (CAPOMF) for Android. In: 2014 International Conference on Advances in Engineering & Technology Research. IEEE, Unnao, Indie, s. 1-7, 2014. Online: [http://dx.doi.org/10.1109/ICAETR.2014.7012912]

Requirements for the semestral defence:

- Items 1, 2 and work started on item 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Rychlý Marek, RNDr., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	November 1, 2020
Submission deadline:	July 30, 2021
Approval date:	February 12, 2021

Abstract

This thesis deals with creation of context-aware mobile application that selects and recommends songs of the music player according to the current context of the device. The device context is obtained from the measured values, which are acquired from the integrated device sensors and from other device system values. The selection of the particular song is based on a machine learning model solution, that classifies the context based on the current data and selects the appropriate song belonging to that context.

Abstrakt

Tato práce pojednává o vytvoření mobilní aplikace zvažující kontext zařízení, která vybírá a doporučuje hudební skladby dle aktuálního stavu kontextu zařízení. Kontext je získáván na základě naměřených hodnot, které jsou získány z vestavěných senzorů mobilního zařízení a z ostatních systémových hodnot zařízení. Výběr konkrétní skladby je poté založen na výstupu modelu strojového učení, který klasifikuje kontext na základě aktuálních získaných dat a následně zvolí skladbu připadající k danému kontextu.

Keywords

mobile application, context awareness, android, mobile device, device context, music player, multimedia application, machine learning, Random Forest, automation, Weka, song prediction

Klíčová slova

mobilní aplikace, závislost na kontextu, android, mobilní zařízení, kontext zařízení, přehrávač hudby, multimediální aplikace, strojové učení, Random Forest, automatizace, Weka, predikce skladeb

Reference

CHMELAŘOVÁ, Gabriela. *Android Music Player with the Song Selection by a Device Context.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor RNDr. Marek Rychlý, Ph.D.

Rozšířený abstrakt

Mobilní aplikace jsou v dnešní době využívány vysokým procentem společnosti. Velká část uživatelů mobilních zařízení je využívá při spoustě každodenních aktivit, jako je práce, sport, relaxování, zábava. Při každé z těchto aktivit často také uživatelé poslouchají hudbu, která může být spouštěna z aplikací na zařízení. Samotná každodenní aktivita může být charakterizována různými faktory, například zda má uživatel mobilní zařízení položené na stole, zda je připojen na WiFi nebo kde se momentálně nachází. Zároveň v dnešní době roste množství mobilních aplikací, které využívají výhod strojového učení pro doručení lepší funkcionality aplikace nebo inovativních nových typů nástrojů. Nabízí se tedy možnost zkombinovat dvě předchozí skutečnosti a vytvořit produkt, který uživatelům pouštějících si skladby z telefonu umožní automatizaci této aktivity.

Následující práce pojednává o vytvoření mobilní aplikace nazvané Context Player, která slouží jako hudební přehrávač s automatickým doporučováním skladeb. Záměrem této aplikace je poskytnout chytrý nástroj, který usnadní uživateli přehrávání skladeb. Typický případ využití aplikace je po jejím natrénování začít vykonávat obvyklou aktivitu a aplikace sama doporučí skladbu obvykle přehrávanou v daném kontextu. Aplikace nejprve načte hudbu z lokálního úložiště, poté při přehrávání skladeb zaznamenává údaje o současném kontextu zařízení a následně po přehrání minimálně dvou skladeb začne predikovat skladby pro daný kontext.

Při zaznamenávání kontextu je sbíráno několik odlišných faktorů a hodnot. Bylo třeba získat obecné povědomí o tom, v jaké činnosti se uživatel právě nachází, tedy jaký kontext se odehrává. Zároveň bylo třeba docílit, aby nebylo třeba volit kontext ručně, ale aby byl detekován automaticky. Toho bylo docíleno sbíráním odlišných hodnot reprezezentujících kontext. Hodnoty sbírané při zaznamenávání kontextu je možné dělit do tří skupin — naměřené hodnoty fyzických senzorů zařízení, odvozené proměnné ze získaných hodnot senzorů a konstanty vyjadřující nějaký stav týkající se zařízení, například připojení k internetu, chůze, připojení sluchátek. Jako hodnoty senzorů byly měřeny hodnoty obvykle se vyskytujících na zařízení jako je sensor okolního světla, vlhkostní sensor či akcelerometr, na druhou stranu jsou zaznamenávány hodnoty i senzorů jako je například sensor srdečního tepu. Naměřené hodnoty senzorů jsou ukládány do soubotu typu CSV a následně dále zpracovávány.

Hlavní částí aplikace je samotný predikační model. Pro klasifikaci kontextů a tedy skladeb které do případného kontextu patří byl využit klasifikační algoritmus pro učení s učitelem Random Forest. Random Forest je klasifikační algoritmus který pracuje dobře nejen s numerickými hodnotami, kterými jsou v našem případě naměřené hodnoty senzorů, ale také s nominálními hodnotami, mezi které patří například aktuálně prováděná aktivita.

Model je natrénován z nasbíraných kontextových dat, které jsou ukládány společně s aktuálně přehrávanou skladbou v daném kontextu. Data vstupující do modelu strojového učení jsou převedena do vhodné reprezentace pro použití v modelu. Model je uložen lokálně na zařízení, proto aplikace nevyžaduje připojení k internetu. Tento fakt je také výhodou z důvodu ukládání dat, které je bezpečnější neposílat přes internet, jako je například součástí kontextu ukládaná aktuální lokace uživatele.

Následné spuštění predikce je podmíněno změnou kontextového faktoru, tedy například připojením sluchátek k telefonu. Vstupem do predikce jsou aktuálně naměřená kontextová data. Po realizaci predikce se náhodně vybere jedna skladba ze skupiny predikovaných skladeb pro identifikovaný kontext. Skladba je následně doporučena uživateli formou zobrazení notifikace. Uživatel může následně z notifikace predikovanou skladbu přehrát.

Implementované řešení aplikace je zveřejněno jako open source na platformě GitHub a zároveň produkční verze této aplikace je zveřejněna na obchodě Google Play. Po implementaci byl model evaluován standartními metrikami pro evaluaci klasifikačních modelů. Spolehlivost predikčního modelu byla testována pomocí vyhodnocovacích algoritmů a později v rámci uživatelského testování několika uživateli testovací verze. Výsledné predikce byly zhodnoceny v kapitole věnující se testování.

Android Music Player with the Song Selection by a Device Context

Declaration

I hereby declare that this master's thesis was prepared as an original work by the author under the supervision of RNDr. Marek Rychlý Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

....
Gabriela Chmelařová
August 1, 2021

Acknowledgements

My sincere gratitude goes to my supervisor, RNDr. Marek Rychlý Ph.D., for his guidance, convenient pieces of advice and continuous support during the work on this thesis.

Contents

Chapter 1

Introduction

Today, the use of mobile devices is high. These devices have become part of our daily activities. We use them for work, relaxation, entertainment, communication or everyday tasks. In the mentioned tasks, mobile devices often come up with a wide range of many different uses. For these activities, it is desirable to reduce the operating time spent on unnecessary repetitive tasks, and thus increase efficiency. To achieve this goal, it is important to identify a recurring task and find a way to automate it. One suitable solution is to use the context of the device.

The goal of the thesis is to find the context of a mobile device to automate operations in a mobile application. In this case, the action will be to automatically recommend the appropriate song in the music player, which means that a specific song will be selected according to the current context.

To detect the context of a device, various information can be used. One of them is to gather the information from the sensors, that the device includes. Every mobile device contains a wide range of sensors. There are about ten different sensors in current mobile phones. The sensors measure different types of values from the environment or from the device itself. An example of values measured on the device may be the orientation of the device, or detection of whether the device is in motion, etc. Examples of values measured from the environment are the location of the device, barometric pressure, etc.

After we get the context of the device, we have to express it somehow so that we can react to it. This can be achieved by specifying the machine learning problem. Machine learning can help classify specific contexts and elicit different responses based on them.

The thesis structure is described in this paragraph. Chapter 2 describes the possibilities of creating a multimedia application for the Android operating system, with possible ways of context detection and the state-of-the-art context-aware mobile applications. Chapter 3 deals with machine learning approaches, which can be used in mobile applications to integrate contextual information. Chapter 4 describes a specification and design of the Context Aware Music Player application. In Chapter 5 the implemented solution of the application is outlined. The final Chapter 6 includes testing, evaluation and discussion of the application prediction functionality.

Chapter 2

Multimedia applications and methods of context detection

Today's mobile devices are often used for many everyday activities. One of the common activities is playing multimedia content, such as music or videos. This feature is achieved through multimedia applications installed on the device. This chapter discusses the development of music player applications that use the context of the device. To cause automatic behaviour of the application a context of the device is used. The context specification is described in detail in section 2.1. a more detailed description of mobile application development can be found in the section 2.3.

2.1 Context detection

It is first necessary to define what a context of the device is. There are several definitions of the expression. One states that context is a real-time entity or information that affects the interaction between a mobile device, a user and an application and it can potentially change the behaviour of a mobile application. The entity can be a user, time, place, activity or object. [46]

What is the context

Context can be categorized by many parameters. One of the possible categorizations is by the source of contextual information, into groups such as user context, environment and physical context, and computing context [43]. Another way is to divide context into two groups — primary and secondary context, according to the effect to the context-aware system. Next context specification also divides context into two categories — active context which directly affects the behavior of the context-aware system and passive context which does not affect the context-aware system. Context categorization can also be divided into three groups, namely environmental context information, application utilization and sensor-based context. The categories of the mobile device context to be considered further will be divided into five groups. The groups are following:

- user context — user preferences, personal time schedule
- device context — operating system, screen orientation
- activity context — specification of the task, time spent on a task or activity

- network context — network connectivity, network bandwidth
- external physical environment context — lighting, temperature, noise level

Not every contextual information may be useful for a specific mobile application. It may be just a few contextual information that will be relevant to a particular task in the application. Information can become relevant if it helps define a specific context. [43]

Context is a dynamic representation of the current state. It is therefore necessary to find a way to represent it as information and to process it dynamically. It then needs to be translated into information that can act upon a specific task. The collected data must then be aggregated, interpreted and stored in order to appropriate action can be taken. [43]

Contextual awareness

Contextual awareness in a mobile application can be specified as the potential of the application to detect and process its context and the ability to adapt the behaviour of the application to provide the user with any useful function. This feature enables the uninterrupted behavior of the mobile application. Using this feature will create a mobile application that will be able to respond dynamically to changing contexts and intelligently help the user. [43]

There are three main ways to obtain contextual data. It involves reading data from various sensors, obtaining system information from a software agent and obtaining data about a specific user. The main way to obtain the majority of the mentioned types of context data are sensors. The sensors can be divided into three main categories — physical, virtual and logical sensors. Physical sensors provide the values for the variables from the environment or from physical aspects of the mobile device, for example, the location of the device given by GPS or device orientation. Virtual sensors provide information about activities executed on a mobile device, operating system, wireless network (such as a Wi-Fi connection to a specific network) or specific applications or software that provides this type of information, for example, calendar events, alarms, etc. Logical sensors are a combination of physical and virtual sensors. They can provide intelligent scanning based on, for example, detecting the current location of the device and if it detects your home location and the ringtone is in silent mode, it switches to normal mode. [43]

The problem can occur when processing different sensor values, as they may have different infrastructures, leading to many different ways of obtaining data from various sensors. Another complication can be noise in the data and the problem of their correct interpretation. This leads to a solution for reading data from different sensors to ensure that the correct context is detected.

Physical sensors

Sensors are tools for detecting the current state of the device. There are many types, in the order of dozens, of sensors in mobile devices. They provide information about the position, amount of light, movement of the device, etc. These values together can help to represent the current state of the device.

Ambient light sensor

This sensor detects the amount of ambient light around the mobile device. This feature automatically adjusts screen brightness, to automatic white balance and helps prevent misoperations when a mobile device is in the pocket. [29]

Proximity sensor

A proximity sensor is used to detect nearby objects. It helps, for example, during a call, when it automatically turns off the display when it detects that it is near the ear. Together with the ambient light sensor, it helps prevent misoperations.

Accelerometer (gravity sensor)

The accelerometer detects the orientation of the device. This causes automatic switching between landscape and portrait mode. This feature is often used in motion-controlled games.

Gyroscope

The gyroscope is used for navigation in space. It measures the orientation of the device and helps to ensure stability of the device. It can be used for navigation when GPS service is not available. It works by measuring the angles from which it derives the current orientation. [11]

Compass

This sensor helps to locate the device more accurately and also serves as a compass itself. This sensor is not included with some mobile devices. It is usually created from a sensor called magnetometer. It indicates the orientation of the device according to the magnetic field. It shows the direction to North and thus it can therefore be used, for example, to automatically rotate the map.

Hall effect sensor

The Hall effect sensor is used to lock the screen when the flip cover is closed on the mobile device, or to unlock it when it is open due to a small magnet contained in the cover. It is able to detect a magnetic field.

Barometer

The barometer sensor helps to obtain the exact height when the mobile device is, for example, in a tall building or under an overpass. It can also be used in mobile fitness applications to detect floor climbing. This can be useful for a more precise orientation in the building, for example when searching for a specific place by floor. [25]

Biometric sensors

Biometric sensors are used to identify the owner of a mobile device, and thus serve to secure the device. For example, there is a fingerprint reader that is used to identify a person using a fingerprint reader. [25]

Ambient temperature sensor

This sensor is used to measure the ambient temperature of the device. This type of sensor is not always available in the device. Before obtaining the sensor values, it is necessary to check whether the sensor is in the device.

2.2 Android operating system

Android is the mobile operating system that is the second most used on mobile devices. It is based on the Linux kernel. The first use of the Android operating system was in 2008. Android applications can be developed in the Android Studio and later published in the Google Play store. It is also possible to install an unofficial application on the Android operating system, but it is a more complicated way because the user needs to download the APK file, that contains the application, manually and grant permission to install the application from an unknown source.

The main language in which Android applications are developed is Java[1] language. Nowadays, a new programming language called Kotlin is starting to being used in Android mobile application development.

Google has introduced an Android-oriented design language called material design, which provides guidelines for an intuitive graphical user interface (GUI). It provides features that mimic real world objects combined with 3D effects and realistic lightning.

Google Play

Google Play is an Android application store, where the official Android applications are published. There are several restrictions on a developer who wants to publish their application. This can be a problem when publishing context-sensitive applications, which often need many permissions to access all possible information about the current state of the device.

Any developer can publish on this platform for an initial fee of 25 USD. Then it is important to create a presentation of the application in the form of a description of the application, entering application parameters, providing screenshots of the user interface or possibly a presentation video. Then it is possible to choose whether the application will be free or paid, and in the case of the paid one, whether it will be a one-time payment or whether it will be paid monthly.

Android Studio

Android Studio is an official IDE for Android applications. It is developed by JetBrains and Google. It is based on the IntelliJ IDEA, which is another IDE from JetBrains.

For application building, Gradle build system is used. It also provides a mobile device emulator for running and testing. The emulator provides a realistic simulation of a mobile device that offers many settings, from default mobile phone settings to mobile device environment settings, such as the physical location of the mobile device, etc. Android Studio also enables various debugging options. [19]

[1] https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html

Kotlin programming language

Kotlin is a new programming language that can be used to develop mobile applications for Android. It is an alternative to Java, which is the main programming language for developing mobile applications for Android. The Android Studio IDE gives users the ability to transfer their old code from Java to Kotlin, or in case that a file is written in Kotlin (indicated by `.kt` extension), then if the developer inserts Java code into this file, it offers the user automatic conversion to the Kotlin code.

2.3 Multimedia and context-aware application development

This feature is achieved through multimedia applications that are usually pre-installed on the device or can be downloaded from application stores. These applications often provide basic functions, which include listing local or online multimedia content, playing and pausing specific multimedia files, or adding them to groups. Due to the frequent use of this type of application, it is desirable to introduce a method of automation into them.

Android multimedia applications

There are tools for developing a multimedia application in the Android operating system that facilitate the development process. For that purpose, `MediaPlayer` API was developed. This API makes it easy to play multimedia content, such as audio, video, or images, in the Android mobile app. Multimedia data that is read from the application can be stored locally on the device or can be retrieved from the network, for example in the form of a stream. [18]

There are two classes that can be used to play multimedia content. First is `MediaPlayer` class, which can load, decode and play multimedia files with minimal overhead. The files can be retrieved from the device, or accessed through an internal URI or an external URL, in the form of a stream. Android supports different types of media files, all listed on the Android developer page[2].

The second class is `AudioManager` class, which handles source audio files and audio output options such as volume level or ring mode. This class also has a method called `isMusicActive()` which returns a boolean value if music is currently being played on the device.

Context awareness on Android

There are many ways to determine the context described in Section 2.1. Specifically, the Android environment provides classes for working with the device context. There are also classes that provide reading data from sensors. The first class to mention is the `android.context.Context` class, where a lot of contextual information is represented through the interface. It issues access to system services that can be used to read sensor data and service status.

Another class is `android.hardware.SensorManager`, which is used to access the device sensors. a list of available sensors can be displayed in this class. The number and types of sensors depend on the version of Android. The available sensors are divided into three groups: motion sensors that measure rotational and acceleration forces (accelerometer,

[2] `https://developer.android.com/guide/topics/media/media-formats`

gravity sensor, gyroscope and rotational vector sensors), environment sensors that measure various environmental variables (ambient air temperature and pressure, lighting and humidity, there are also barometers, photometers and thermometers) and position sensors that measure the physical position of the device (orientation sensors and magnetometers). [14] Most sensors are available on the majority of today's devices, but some more specific sensors may only be included with a few mobile devices. [30]

Besides physical context, the `Context` interface also provides the computing context of the device. This function provides the `android.net.ConnectivityManager` class. ConnectivityManager class provides information about the state of the network connectivity and notifies the applications of a change in network connectivity. The network connection can be Wi-Fi, GPRS or UMTS. [17]

Wi-Fi and other features are provided through the `WifiManager` class. The same can be found for Bluetooth via `android.bluetooth`, which provides features such as scanning other Bluetooth devices, searching for paired Bluetooth devices, etc. [20]

To obtain current audio or visual information about the device, classes `DisplayManager` for information about the display and `AudioManager` for information and ability to manipulate the volume and ringer mode, can be used.

To obtain information about device location services, `LocationManager` is used. Location information can be provided by more than one source, such as Wi-Fi or GPS. They differ in accuracy, speed and battery usage. [30]

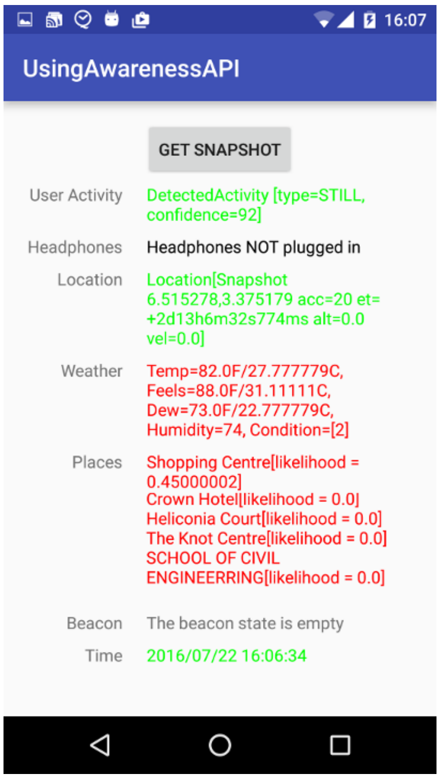
The rest of the computing context can be obtained using the Settings API or access to the underlying Linux operating system. `Settings.System` provides access to information such as the current screen brightness or ringtone volume. Likewise, `Settings.Global` and `Settings.Secure` provide system settings information that can only be read but not modified. [30] To get the information about the running processes, the class `Process` can be used.

Awareness API

Provided by Google, this API can be used to work with the user context. It was first introduced at Google I/O 2016 and specifies users in specific situations. It is possible to monitor the current state of the device or create a conditions called virtual fences, that will monitor the situation and notify the application by the operating system. The Awareness API provides seven default context types. These are the local time, location including longitude, latitude and elevation of the device, user activity such as walking, running or in the vehicle, headphone state, if they are plugged or not, weather conditions, place description and information about nearby beacons. Because the Awareness API is a part of Google Play Services, the application has to be set to use Google Mobile Services. [38]

The above context information can be obtained using the previously mentioned APIs, but the advantage of using the Awareness API is that it is easier to retrieve data, the data are already processed by the operating system, so it is not raw data, it is accurate and the API is optimized to not drain the battery too much.

There are currently two types of uses for the Awareness API. First is the Snapshot API, which takes a snapshot of the current context, which can be seen in Figure 2.1. The second is the Fence API, which defines a set of possible conditions, and when the operating system detects that all conditions are met, it notifies the application.


16:07

UsingAwarenessAPI

GET SNAPSHOT

User Activity	DetectedActivity [type=STILL, confidence=92]
Headphones	Headphones NOT plugged in
Location	Location[Snapshot 6.515278,3.375179 acc=20 et=+2d13h6m32s774ms alt=0.0 vel=0.0]
Weather	Temp=82.0F/27.777779C, Feels=88.0F/31.11111C, Dew=73.0F/22.777779C, Humidity=74, Condition=[2]
Places	Shopping Centre[likelihood = 0.45000002] Crown Hotel[likelihood = 0.0] Heliconia Court[likelihood = 0.0] The Knot Centre[likelihood = 0.0] SCHOOL OF CIVIL ENGINEERRING[likelihood = 0.0]
Beacon	The beacon state is empty
Time	2016/07/22 16:06:34

Figure 2.1: Awareness API — taking snapshot of the current context screen (Source: [39]).

AWARE

AWARE is a framework used to derive, log, and share contextual information obtained by sensors. AWARE records hardware, software and human-based data, which are then analyzed using AWARE plugins. It's easy for individual users to try, because AWARE has an application that allows the user to enable or disable reading from different sensors, as well as to enable or disable specific plugins. The data are stored locally on the device or if required on a remote MySQL server, and then the AWARE dashboard can be used. AWARE does not store any personal data, such as telephone numbers or contacts. Additional add-ons can be installed in the application to improve the functionality of the context specification. [13]

AWARE also provides the ability to scan the device where the application is installed and sends reports to the AWARE dashboard. It is also possible to remotely run an application query on a subscriber device.

The AWARE framework also provides the AWARE API, which is available as a library for Android or iOS. In this implementation, it shares the user context at the operating system level.

AWARE is not published to Google Play because it uses many permissions that the current development guidelines do not allow, so AWARE client cannot be published. The AWARE client can be downloaded here[3]. The application is open-source. [6]

Plugins are used to provide abstract data for reading sensors that represent the context. Plugins can reuse data from other plugins to create a more accurate context description.

[3]`https://awareframework.com/`

Plugins also include data mining and machine learning methods, but may not have an interface for users. The application provides an interface for interpreting the context and gaining an overview of the context data. [13]

CAreDroid

CAreDroid is a framework for contextual applications for Android. It allows developers not to have to deal directly with context monitoring and customization in application code. Within this framework, users provide application logic in the form of a list of methods that will respond to a specific context, and a number of permissible operations are set in these specific contexts. CAreDroid monitors the context of the physical environment and manages context-sensitive methods, which means that it only activates specific blocks of methods that are the most appropriate for the current context.

CAreDroid provides direct reading of the raw hardware context, such as battery level, or a Wi-Fi connection, or a derived context in the form of a mobility status, such as walking or running, which requires further processing of the raw sensor data. [12]

CAreDroid gets the current context with less overhead than the Android Java API. Raw context monitoring is performed on the software stack to the hardware abstraction layer. This layer provides sensor managers that connect low-level drivers and the application. To reduce overhead, it is important to bypass the hardware abstraction layer with sensor managers. In this framework, it is implemented through the VM Dalvik thread.

The derived context can be obtained from calculations and observations performed on raw data. For example, the mobility status is detected by reading the raw accelerometer data and then using the classification to detect the current mobility status.

The CAreDroid adaptation engine helps distinguish which method to choose by using a conflict resolution mechanism that chooses the method with the highest priority. This is the principle of replacement methods at runtime. a configuration file is used to find the best method.

CAPOMF

Context-aware and pattern-oriented machine learning framework (CAPOMF) are a proposed guidelines to build robust applications on Android with context-aware services. The core element of such a system is called *agent*. Agent is a component with which the application communicates to which handles context-aware services [41]. The agent collects all data from services provided by CAPOMF and applies the learning algorithms on them in order to make a prediction.

This lowers the amount of involved interactions of user and sensing services. The proposed types of data collected through sensing services are user location, weather information, the energy level of the device and accelerometer readings to determine shifts in the position of the device. The next proposed idea is to use push notifications to provide the user with recommender system preferences.

EgoSENSE

EgoSENSE is a context-aware mobile framework to help with the implementation of contextual services. This framework collects data from sensors, processes and justifies events, and stores them in the device. Framework layers are independent and can be changed to respond to the user's needs. The first layer obtains data from the physical sensor from the sensor provider. To obtain the data from a sensors, the Funf framework described later in this section was used.

For the purpose of demonstrating the framework, the Personal Health Monitoring application was created. After testing the application, it was assumed that the application and the framework used did not have a significant effect on energy consumption. Data from the sensors is processed locally on the device, which helped significantly reduce network traffic. Applying a filter to contextual events helps reduce the amount of data published to the cloud. The application only sends notifications to the cloud that contain the result of the sensor analysis. [36]

Funf

Funf is a framework that allows the collection and analysis of mobile data. The main idea of this framework is to collect the data from different activities, analyze them and find patterns in them. Funf's core function is to serve as a „probe", which means a software module that is collecting data from physical sensors such as gyroscope, accelerometer, etc., used to monitor higher level activities.

The framework has about 30 default built-in probes that serve as detectors of device motion, position and its environment or detectors of the interaction of the device, such as applications running in the background, current activity on the screen and browser bookmarks. Social interactions such as contact information, call logs or SMS are supported too. The basic framework can be further extended to include more monitoring abilities. [5]

2.4 Current context-aware mobile applications

The use of context can be applied to reduce the required user interaction with the mobile application. Today, there are many mobile applications on the market, that use the context of the device in this manner.

Context-aware mobile applications can be classified by various criteria, such as the type of context that is used in the application, the purpose of the application, or by the active or passive reaction of the application. [46]

Context-aware automation tool

This is a group of mobile applications, that the user can use to create an automation of the task on their device based on the specific context. This is a slightly different type of application, as it uses the context of the device, but the user has to manually set up the automation.

Automate

Automate is an Android mobile application that enables the user to create their own context-based automation. Individual contextual parameters are presented as a „building blocks" of the tool. The user can then define (as seen in Figure 2.2) the automation flow. The application provides the possibility to choose more than one context and create dependencies and conditions on them.

The application currently provides 340 various building blocks that can be of different types, for example, camera, alarm, calendar event, monitoring of the device parameter such as CPU speed, data usage, battery usage, if any media is playing, location sensor reading, etc. The application enables predefined options for basic automation. [22]

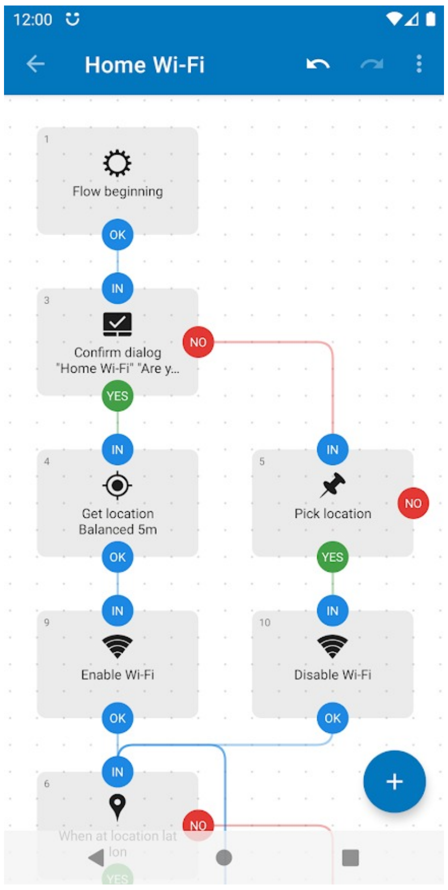


Figure 2.2: Automate application — building blocks (Source: [32]).

Conscient — context-aware application

Conscient allows the user to create their own automation based on their current context. For example, in the application, the user can set the music player to open when headphones are connected, or to open a fitness application when the device detects the person is running and the headphones are connected. These options are presented in cards, that are shown in Figure 2.3. One of the possible contexts, that can be detected, is whether the headphones are plugged in or not, and the type of movement of the person — whether the person is running, is in a car or on a bike.

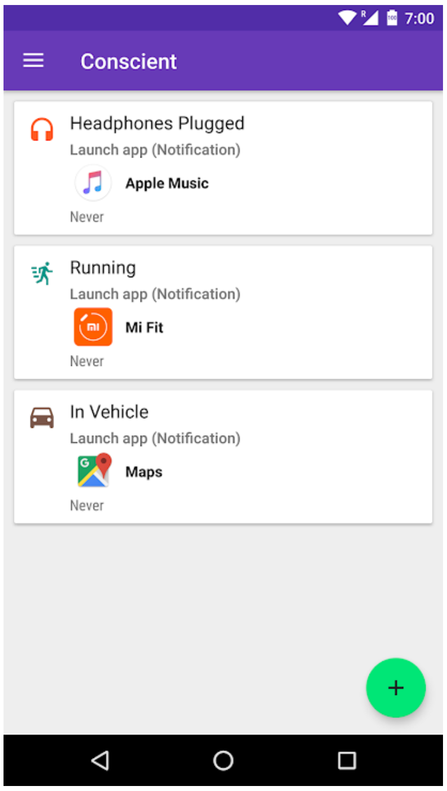


Figure 2.3: Conscient — the overview of automation (Source: [4]).

InCarMusic — context-aware music recommendation in a car

This concept of a context-aware application is based on systems that can adapt to specific situations while the user is traveling by a car. The system is considering the type of the road, traffic conditions or the user's mood. The context is inferred from user ratings and reports in the application. This process is possible to be seen in Figure 2.4. [7]

In this case, the context was represented by the following set of independent contextual factors, listed in the table below.

Contextual factor	**Contextual condition**
driving style	relaxed driving, sport driving
road type	city, highway, serpentine
landscape	coast line, country side, mountains, urban
sleepiness	awake, sleepy
traffic conditions	free road, many cars, traffic jam
mood	active, happy, lazy, sad
weather	cloudy, snowing, sunny, rainy
natural phenomena	day time, morning, night, afternoon

Table 2.1: Table representing a possible contextual factors used in the InCarMusic application. Table source: [7].

For testing purposes, there were five songs used per each of ten genres. The application should work based on the user's recommendation. In order to test the particular model situations in the car, a web application was built, that simulated specific contexts and asked the user to rate the music according to the current situation. For a relevant estimation of the contextual factors, users were asked whether the particular factor (for example sunny weather) has a positive or negative effect in their choices of a specific genre. [7]

In the next phase, the aim was to find out, if the rating depends on the contextual conditions. For evaluation of the dependency of contextual conditions on the music track rating, the Matrix Factorization model was used. The use of this model has shown an improvement in personalised and non-personalised predictions. [7]

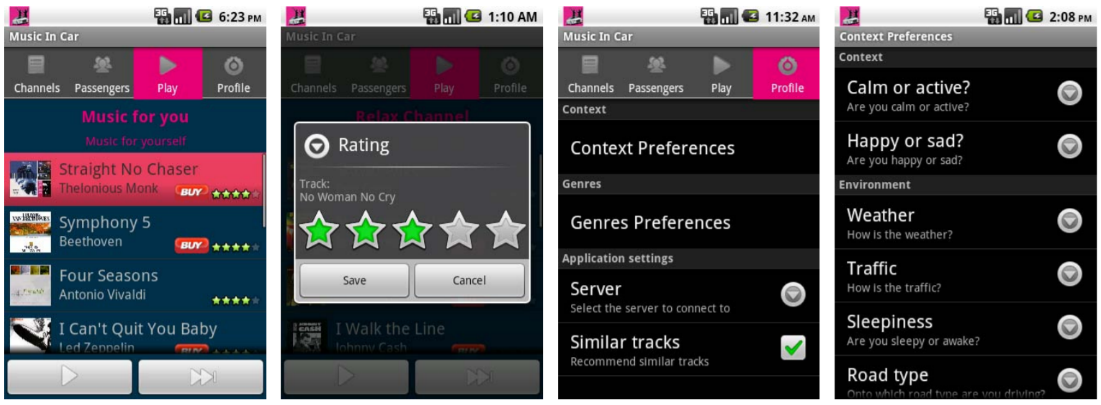


Figure 2.4: From the left, the first screenshot shows tracks that are proposed to be played, the second screen shows a rating process, the third screen shows the user profile preferences connected with context and in the final screen there are options to configure the recommender (Source: [7]).

Context-aware nurse application

The aim of this application is to simplify human machine interactions of nurses with the administrative part of their work and to simplify access to the required information. This application uses contextual information in cooperation with machine learning to create a dynamic user interface that will change according to the situation. [27]

The context of this application was primarily determined by the mobile device's indoor location. For this information, a WiFi positioning or near field communication was considered, with NFC being used in the end. [27]

As the model for determining the current context, the following machine learning method was chosen. The supervised learning method was used, closer described in Section 3.1. The type of the supervised learning method was classification. As a classification approach, a decision tree was selected (described in Section 3.1). [27]

2.5 Summary of contextual awareness approaches

In this chapter, the context of the device was introduced. Along with that, the Android platform was described, together with its tools for developing a context aware application. For that purpose frameworks such as Awareness API, AWARE, CAreDroid, EgoSENSE or Funf were also mentioned. All these frameworks simplify in various ways the development of context aware applications on the Android operating system and can be used further in the implementation part of this thesis.

In the end, the mobile application that use the context of the device to do certain tasks were closer described. The emphasis was put on their way of collecting the contextual data and the way of interaction of the application with the user, based on the detected context.

From the mentioned applications result the following ideas, that can be used in context aware application implementation or aspects that can be avoided. The Automate application provides a wide selection of options for activity automation, which can be really handy in case of a desire to develop a very specific action. On the other hand, some users reviewed the application on Google Play store[4] as too much complicated for the normal user. Thus this fact of giving too many tuning options will be avoided in application design. The Conscient application provides a much easier environment than the Automate application. The inspiration for the context aware application will be the detection of activities such as running, traveling by car or plugging in the headphones. InCarMusic provides a useful idea in case of deciding what are the factors of choosing the particular songs in the given context. This can help to determine which is picking a song potentially dependent and can help in specifying the contexts in the implemented application. The nurse application proposed the solution of using decision trees as a context classification method which works well with both numerical and nominal values. This fact will be later considered in context aware application design.

From the above mentioned frameworks, the CAPOMF suggests a suitable approach in dividing the application into application communicating with an agent which handles the sensor reading and machine learning model processing. Furthermore, there were also mentioned useful types of contextual information to be collected such as device location, battery status or surrounding weather conditions.

[4]https://play.google.com/store/apps/details?id=com.llamalab.automate&reviewId=gp%3AAOqpTOEhRxNS6DJUDscmpzkYCujcvnA_n_7oem5xYFUihSaJK5Szte7vEgfY8lLWn-y9dQ-tV7Agi99UlEVnmHQ

Chapter 3

Machine learning in context-aware applications

To gain the advantage from using the context of a mobile device, it is necessary to find the specific technique to classify it. For this task, machine learning algorithms can be used. This chapter provides an introduction to machine learning principles together with an overview of currently used machine learning techniques. At the end of the chapter, the right algorithm to be used in the context-aware music player application is deduced.

3.1 Machine learning approaches

This section deals with an overview of current machine learning techniques that are currently used to solve artificial intelligence problems.

Machine Learning

Machine learning is a discipline of artificial intelligence that can programmatically learn from input to provide an output. This provides the algorithm with the ability to adapt to the changing input datasets without the input from the user. [28]

Machine learning is a constantly learning activity. Its algorithms process input data, learn from them and then use the obtained knowledge to discover patterns in the data. In the early stages of learning a machine learning algorithm, it may be necessary to correct the way it is learned. The machine learning algorithm learns with time, which means that they are continuously evolving with the new data samples and experiences gained from the data. Today it finds use in online video streaming IoT, self-driving cars or for example transportation and logistics. [23]

Supervised and unsupervised learning

In machine learning, there are two main approaches — supervised and unsupervised learning. The supervised learning gives the algorithm example results of what the predictions should look like. It is a process where the models try to learn the mapping function between the input and the output. For example, the algorithm can have images as input and each image would have a label of what is in the picture. Then after learning the pattern, the model should predict the correct label after being shown a new image. This method is good for consistent data. Two main types of supervised learning are **classification** and **regression**, which are more closely described in Section 3.1.

Another option is unsupervised learning, which does not provide the labels of the input, so the algorithm must decide according to the clustering of example input data. This option is good in the case that there is a wide range of allowable predictions. This method can be used when it is not clear, what the final answer of the model should look like. The two groups of unsupervised learning are **clustering** and **association**. The dataset is given to the model without explicit instructions of what to do with it. This method can be used for example for face recognition. [42]

Unsupervised learning can group data in different ways. There are four types of ordering the data:

- **Clustering** — The first is clustering, which means grouping data based on similar features, for example, a bird pictures that order bird species based on the color of the bird on their beak size.
- **Anomaly detection** — Anomaly detection is another output of unsupervised learning and it serves to find unusual patterns in the data, for example, if a credit card is used in Europe and America within the same day, there raises a suspicion above the unusual behaviour. That is called looking for outliers in the data.
- **Association** — In this type of desired output of the unsupervised learning, there are features of the data, that directly correlate with other features. For example, in online shopping, it can determine if the person is shopping for a woman or a man article. It works on few attributes of the data to predict other commonly associated attributes of other data.
- **Autoencoders** – An autoencoder takes the input data, converts it into a code, and then tries to recreate the original data from the code. This technique can be useful for, for example, removing noise from images, videos or medical scans and thus improving their quality. This can be achieved via training the model on clean or noisy data to learn it to detect and remove the noise later on the input data. [42]

There is also an option called semi-supervised learning, that combines the above options. This principle is based on cases, where it is hard to label all the data, for example, disease from a medical scan, but it is still useful to have a label at least on some of them.

The special type of learning is reinforcement learning. Reinforcement learning is based on looking for a goal in an uncertain environment. The machine learning model does not get the answer what is the desired output, but it gets a reward or penalty according to its actions. The model does at the beginning random choices and it tries to continuously maximize the reward. This type of learning is often used in computer games. The strong side of this approach is trying many possible paths to find the best option. [40]

Classification and regression

Supervised machine learning is divided into two groups, classification and regression. Classification results in putting the data into separate classes according to their nominal label. This label can be for example a type of fruit. Regression results in continuous data analysis, which means that the output is a number, for example, the weight of an object.

Classification can be further split up according to the number of classes, into which it is split. There are three main groups:

- **Binary classification** — in this type of classification there are only two possible output classes, for example, male or female. Linear regression, Naïve Bayes or support vector machine algorithms can be used for this type of classification.
- **Multi-class classification** — the input is classified into more classes. Each input can be in only one class. For multi-class classification can be used naïve bayes, decision tree, random forest, or K-Nearest Neighbors algorithms but also the algorithms mentioned for the binary classification, which can be modified to multi-class classification. This modification can be done by the principle called one-vs-rest, which means that the classification is separated between one class and the rest of the classes.
- **Multi-label classification** — each of the input data can be labeled in more than one class. For example, the newspaper article can belong to sport, entertainment, and location together. For using this type of classification, the multi-label decision trees and multi-label random forests can be used.

Classification algorithms

There are several algorithms in the case of classification. There is an overview of the main algorithms. This section provides a closer description of Naïve Bayes, decision tree learning or support vector machines, random forest, which is an adaptation of many decision trees and K-Nearest neighbors algorithm.

Naïve Bayes

Naïve Bayes classifier is a machine learning model that is used to determine the probability that the particular input belongs to a specific class. The classifier is naïve because it assumes that inputs are independent and they do not influence each other. The advantage of this model is that it is highly scalable and cost-effective.

Decision Tree

Another machine learning classifier is Decision Tree, which can be used for classification or regression. This approach works on the principle which divides input data into groups until it matches or does not match the prediction. Individual tree nodes represent decisions and branches of possible options. End nodes represent particular predictions of the classifier.

Nodes are represented as if-then options which are mutually exclusive. The tree has a top-down structure. Decision tree options should be categorical or discretized. Parameters in the top level of the Decision Tree have a higher impact on the classification.

Random forest

This algorithm is based on high number of decision trees. Each decision tree makes a class prediction, after which the votes are summed up from all decision trees and the result class is the class with the most results of the decision trees.

Support Vector Machines (SVM)

Support Vector Machine is a binary linear classifier, which splits data into two sets along the hyperplane. It is a supervised learning method for classification, regression and it also can be used for outlier detection. It allows to choose different kernel functions for a decision function. It is effective for high dimensional space. It can also be used in cases where there is a higher number of dimensions than the number of inputs. This can also be a disadvantage because it can lead to overfitting, so it is crucial to choose the right kernel function. [44]

K-Nearest Neighbors (KNN)

This algorithm works with the assumption that similar objects have similar attribute values. K-Nearest Neighbors algorithm is used for both classification and regression problems. It is realized by grouping the objects that are close to each other in the graph, which visualizes the objects by their features. The distance is obtained in a form of Euclidean distance, which means a straight line distance between two points.

Clustering and association

Unsupervised learning has two main categories — clustering and association. In clustering, the goal is to find the close groups of the data, for example, customers shopping online with similar behaviour. The goal of the association method is to find a set of rules that describe a portion of the data, for example, people that are shopping online, what they often buy together if they buy a specific item.

3.2 Machine learning on Android

Machine learning methods can be applied to Android mobile applications via libraries. The library is called TensorFlow Lite.

ML Kit

ML Kit is an SDK for mobile devices developed by Google. It is a part of the Firebase platform. It provides on-device machine learning techniques, which allow real-time processing, for Android and iOS. The on-device option provides offline functionality. It provides a set of trained models. ML Kit provides by default two main groups of models — vision and natural language models. In the vision group, there are trained models for text recognition, face or pose detection, barcode scanning, image labeling, etc. Natural language group includes language identification, text translation or creation of smart replies. It is also possible for ML Kit APIs to replace their default model by a custom model trained by using TensorFlow Lite, which is described in detail in Section *TensorFlow Lite*. All models offer offline mode, which allows to use the machine leaning resolution without internet connection. [21]

TensorFlow Lite

TensorFlow is an open-source library from Google which provides an implementation of machine learning for Android. Its lightweight form is called TensorFlow Lite and it is suited for running on mobile devices. It provides on-device machine learning model resolving and running, which result in high speed resolution of ML tasks. It uses hardware acceleration with the use of Machine Learning APIs.

To avoid high GPU processing power usage and long times of training a model, it is possible to retrain the previously created model to create a new model.

Model and data location

After choosing an appropriate machine learning model and the way of integrating it into an application, it is important to choose the location of the input data and the model itself. There are two options — running the model on a device or running it on a server. Today's technologies allow running the model primarily on-device, which is an advantage because of a lower cost, more data privacy, lower latency and no need for internet connection.

Machine learning for context-aware applications

From the mentioned approaches, it is important to determine suitable options for using machine learning in context-aware mobile applications. The suitable algorithm that can be used for context classification was proposed in a number of context studies and context-aware applications. In this chapter, the description is provided on how to choose an appropriate algorithm for context classification.

Specification of the context detection as a machine learning task

The classification of a context of a device can be specified as a machine learning task. The input of the machine learning algorithm is the contextual data that can be obtained from the device's sensors readings and from the virtual device context. The output of classifying the current context is the specific action. Thus, this is a supervised learning task.

3.3 Context-aware application with machine learning

Below, there is an overview of example usage of the machine learning algorithms in context detection. The application in question uses different machine learning algorithms and methods of processing and interpreting the data.

Decision tree in nurse application

The application is the nurse application, described in Section 2.4. In this application the decision tree was used, to predict the user behaviour. For the implementation of the machine learning algorithm, the Scikit-learn[1] python library was used. The algorithm runs on the server and data from the device are sent via a HTTP request, where the server processes them and makes a prediction upon them. The data are split according to the key features in the decision tree. There are five possible classifications of contextual situations, for example, „take patient measurements", „patient activities", etc. The end nodes of the decision tree represent the resulted data clusters. For example, in Figure 3.1, there are two nodes with a *value* array that contains the number of input samples assigned to the specific class, which represents the position in the array.

[1]https://scikit-learn.org/stable/

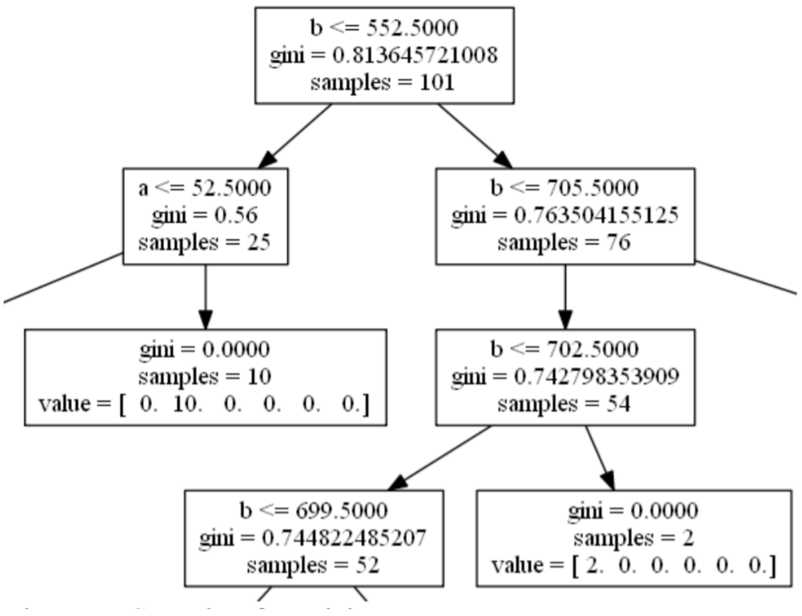


Figure 3.1: The sample of decision tree data structure in nodes of the tree from the nurse application (Source: [27])

3.4 Summary of machine learning methods

The chapter introduced machine learning and its approaches. The suitable approach for context-aware applications is to use a classification. The main classification methods were mentioned in Section 3.2. There were different types of classifiers, the first was binary classification (between two classes) as for example Naïve Bayes classifier or support vector machines, or multi-class classifiers that allowed to separate the data into multiple classes such as decision tree or random forest.

There were also introduced ways of integrating machine learning into Android applications, for which the ML Kit a TensorFlow Lite frameworks were mentioned. At the end of this chapter, a closer look was taken at the context-aware application and its way of using machine learning for context detection. From the mentioned machine learning methods for the classification, the Nav̈e Bayes is effective but its downside is that it assumes that the input variables are not correlated with each other, which would be a problem because the different samples in future context aware application can be bound to the same context. K-Nearest neighbors have a disadvantage in the obligation to choose the number of classes beforehand, but in case of future application this variable can be unknown. The SVM is a promising algorithm but to be able to classify nominal values, they need to be converted via one-hot encoding. Thus the last option, the decision tree or its variant random forest are the most promising algorithms. They natively handle nominal values without the need of conversion, the number of output classes does not have to be specified in advance and they are well scalable with a large amount of data.

Chapter 4

Specification and design of the application and model

This chapter deals with the description of the specification and design of the application and machine learning model. The chosen target platform is the Android operating system, which was more closely described in Section 2.2. The formulation of the application's typical use is described in Section 4.1. Section 4.2 describes the specification of individual application entities. In Section 4.3, the design of the application user interface is described. Section 4.4 deals with the specification of the machine learning algorithm used in the application.

4.1 Typical use of the application

The application will allow everyday music listeners to automate their habits for playing music. The application will provide automatic recommendations of songs according to the specific context of the device. a typical situation will be when the user is performing a normal activity, such as running, an afternoon relaxation song, or concentrating while at work. Once the user is in the usual context, the song or playlist will be suggested in the notification bar. The machine learning model of the application will be trained from the initial manual selection of songs in different contexts. The training phase will last throughout the life of the application, each time a song is selected manually, the model adapts.

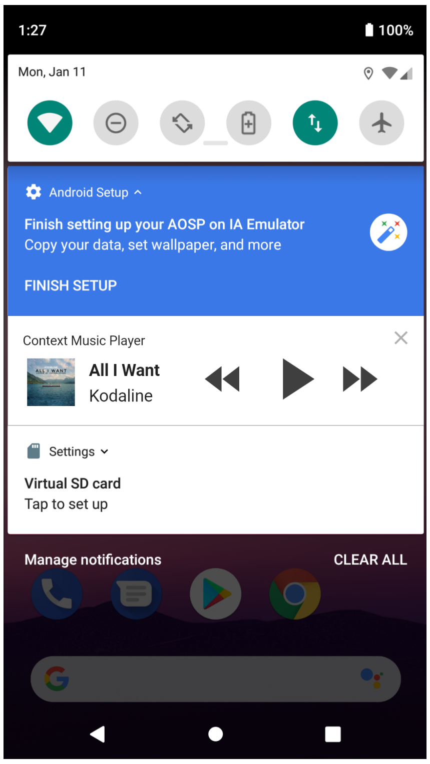
4.2 Application specification

This section lists the application specifications. The main functions of the application, key parts of the machine learning model and the user interface of the application are mentioned here. This section mentions the use of an open source music player, the source and storage of data, the chosen machine learning model and its location.

Specification of the user interface

The core part of the application is a music player. The main subject of the application should be a music player. When the user opens the application, he should see the playlist. The user interface should be simple and logical. It should offer basic functionality in the form of the ability to play the desired song. Music files should be loaded from local storage and displayed on the app's home screen. This proposal is based on the assumption that application users will have songs stored locally on the device.

The second main function of the application will be the song prediction. The prediction should be shown via the notifications in the notification bar. The notification should allow to play the predicted song straight from it. The first concept of this component can be seen in Figure 4.1. Buttons for skipping the recommended song can later be used to modify the model's learning. The notification should provide the main information about the song, title and author, and control buttons for the song, such as play or pause the playback.


1:27 100%

Mon, Jan 11

Android Setup

Finish setting up your AOSP on IA Emulator
Copy your data, set wallpaper, and more

FINISH SETUP

Context Music Player

All I Want
Kodaline

Settings

Virtual SD card
Tap to set up

Manage notifications CLEAR ALL

Figure 4.1: Proposed notification of the music player when a context situation is detected.

Specification of application features

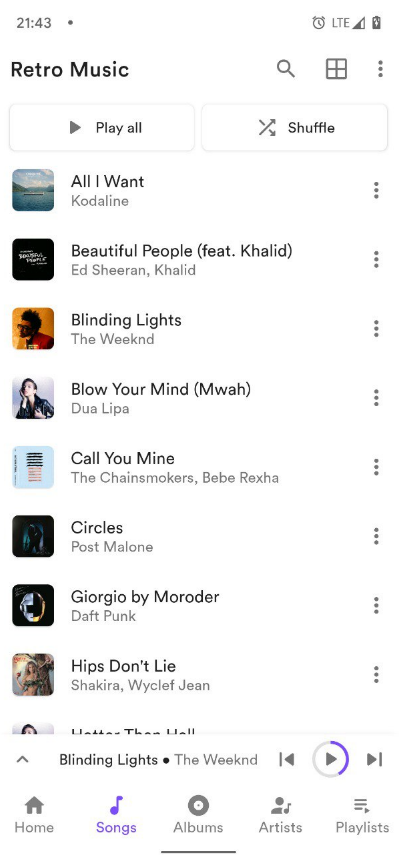
The application should provide the following features. First, it should be possible to manually select the desired song. After learning the device context, the application should provide recommendations based on the user's context. Such recommendations can be made through notifications in the notification bar, where it should be possible to press the play button on the notification and automatically play the selected song.

The application should then automatically recognize the situation to play music from the contextual information and further distinguish the appropriate song for the current situation. For this function, it is necessary to use the machine learning model described in Section 3.1.

The application will also require a high number of permissions to grant. The permission request will be processed according to the Android permission instructions.

Music player

As a music playback component, the open source music player called Retro Music Player was proposed. It is a customizable complex music player which provides an easy-to-understand material design user interface. The player offers many functions, in addition to the basic ones also for creating, editing or importing a playlists. It is also possible to add or edit song tags. The application was chosen for the functionality of creating playlists and for the clean look that can be seen in Figure 4.2.



21:43

Retro Music

Play all | Shuffle

All I Want
Kodaline

Beautiful People (feat. Khalid)
Ed Sheeran, Khalid

Blinding Lights
The Weeknd

Blow Your Mind (Mwah)
Dua Lipa

Call You Mine
The Chainsmokers, Bebe Rexha

Circles
Post Malone

Giorgio by Moroder
Daft Punk

Hips Don't Lie
Shakira, Wyclef Jean

Blinding Lights • The Weeknd

Figure 4.2: The user interface of Retro Music Player showing the list of songs (Source: [26]).

4.3 User Interface design

This section describes the visual and presentation parts of the application, including the name of the application, the icon, and components of its user interface. The application has to compete with modern music applications, and therefore must attract the attention of a potential user through an external presentation. Emphasis was placed on the main use of the application, i.e. playing songs both manually and with the help of prediction. Furthermore, emphasis was placed on the simplicity of appearance, but also on maintaining the appearance of a state-of-the-art music player to make its controls clear.

Application name

The application name was chosen to *Context Player*, which is an explanation of its two main functions — standard music player, which captures the current context of the device and works with it. When choosing an app name, it was important that it be short so that it was easy to read in the app menu of your Android device. Therefore, the original names in the description of — *Context-aware Player* or *Context Music Player* could not be used because the name in the menu would be abbreviated.

Application icon

Another essential part of the presentation of a mobile application is its icon. It is important to visualize the purpose of the application in the form of a self-describing icon while maintaining the Material Design rules that specify the properties of the icon. The Material Designs instructions describe exactly the rules that an Android application icon should follow.

Each icon should have the correct proportions and shape to ensure that it appears well on all devices later. Then it is important to consider layering the icon, that is, if the icon has a colored background or frame different from the icon image. Another important part is the shadows and simulated lightning of icons. The dropped shadow or light should be similar to other applications, so it must follow the established rules. State-of-the-art rules also say that the creation of three-dimensional objects should be avoided. Icon images can have shades that mimic the third dimension, but should be done in a more flat, material style. An icon that complies with these rules should ensure a consistent look with the other applications on the menu. This also leads to a higher chance of downloading, using and leaving the application installed on the device.

The Context Player application icon consists of two layers — a dark background and a pink logo in the foreground. The dark background illustrates the dark theme of the application. The foreground logo displays a play button similar to the button used in the application to easily recall in the user what is the main function of the application. The colors of the logo are similar to the primary color of the application, which is used on all screens throughout the application.

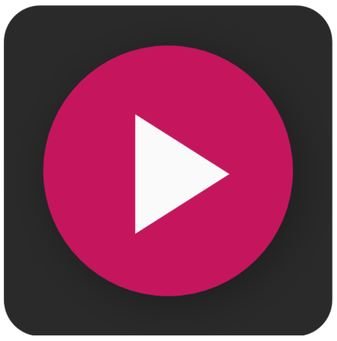


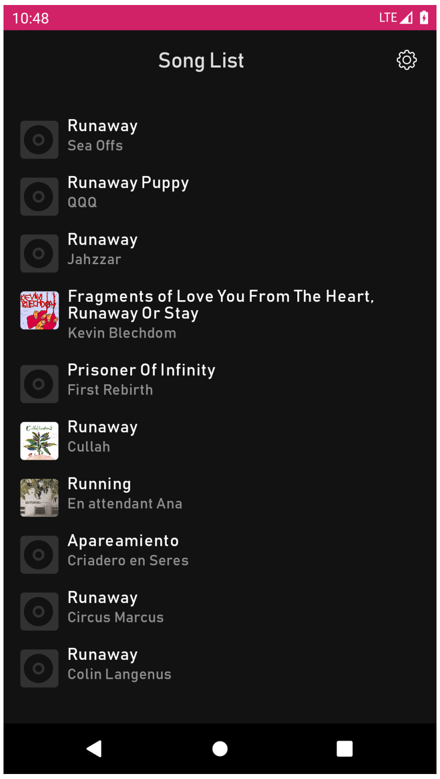
Figure 4.3: Context Player launcher icon.

The icon was designed using Affinity Designer for vector graphics and exported to the required 512 px x 512 px PNG format for later use in the application. Loading an icon into an application in Android Studio is a straightforward action that involves loading an icon using a dialog box that shows the various possible icon sizes and also resizes the icon to fit all possible shapes of other icons on each device (round or rectangular). Then the icon is loaded in all possible resolutions into the */res/mipmap* folder of the Android project and is automatically set as the main application icon.

Application user interface

The application consists of three main visual parts — the playlist, the screen of the currently playing song and the settings used to obtain information about the context detection. The original mock-up was created using Adobe XD. Application icons were downloaded from the official Material Design icon page, the free icon page, and the main play icon was designed individually using Affinity Designer. The play icon was then converted to SVG vector format to be accepted as an Android Studio application icon.

The first screen that appears when the application is opened is the playlist screen. The songs are sorted by the order in the local music folder from which they are loaded, which is often related to the date the item was added. The emphasis in the text of individual items is placed on the name of the song, which is the main identifying element used in other music applications. The name of the current playlist appears in the top application bar. In the upper right corner is the settings icon, which is visible from all other screens and which leads to the settings screen, that is described later. The decomposition of individual items, such as spacing between items, is guided by material design guidelines. It is also possible to refresh the playlist by dragging the window down, in case a new song has been downloaded or an old song file deleted. Clicking on each song item leads to another screen of the application which shows the player itself. The layout of the playlist screen can be seen in Figure 4.4.


10:48 LTE

Song List

Runaway
Sea Offs

Runaway Puppy
QQQ

Runaway
Jahzzar

Fragments of Love You From The Heart, Runaway Or Stay
Kevin Blechdom

Prisoner Of Infinity
First Rebirth

Runaway
Cullah

Running
En attendant Ana

Apareamiento
Criadero en Seres

Runaway
Circus Marcus

Runaway
Colin Langenus

Figure 4.4: The playlist screen designed for Context Player application.

The second main screen of the Context Player application is a screen with the playback control. The player screen is a simple version of a typical music player. As it is possible to see in Figure 4.5, it displays typical song information, such as the title of the song being played, its author, and the album art. Below the song's author is a progress bar that tracks the current position of the song. The player screen also provides access to the settings screen and the back button used for navigation.

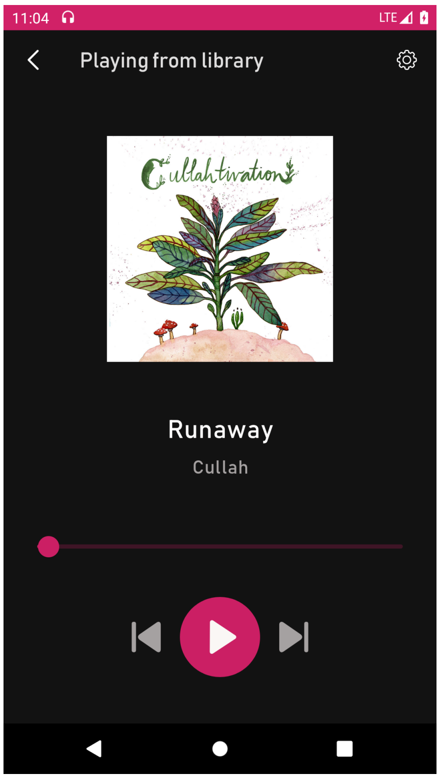
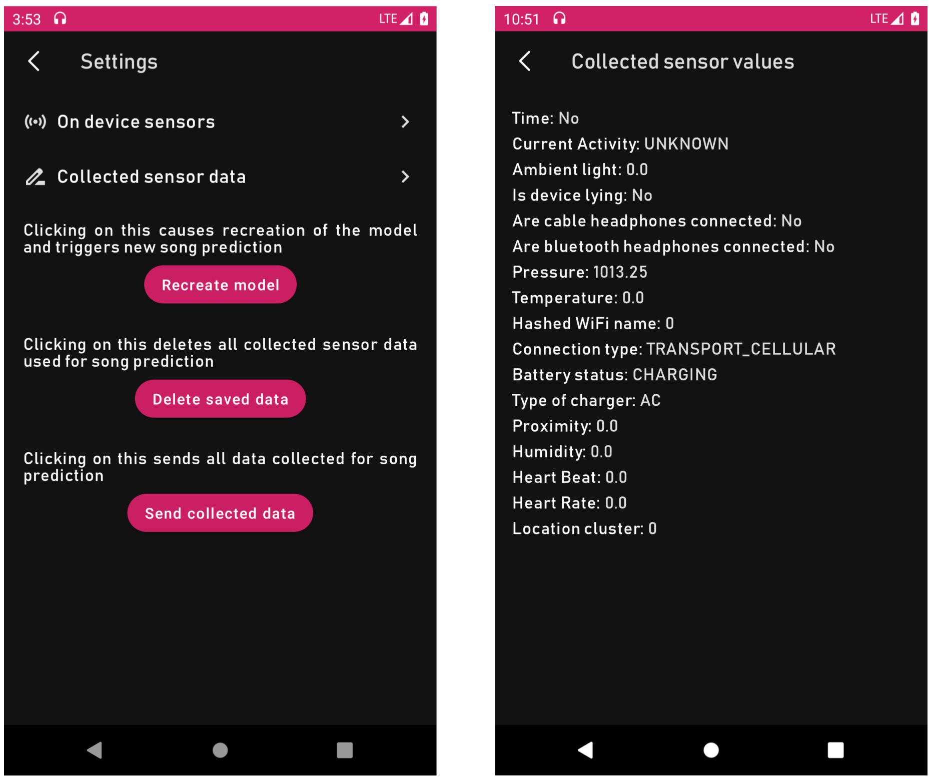


Figure 4.5: Context Player main playback screen, which displays information about the currently played song and allows the playback control.

The settings screen provides information about the sensors and allows to manage the prediction and collected input data. As shown in Figure 4.6, it contains two navigation buttons and three control buttons. The navigation buttons lead to two more screens with sensor information. The first screen that can be accessed contains a list of available sensors on the device. The second screen displays all data with their current values that are used to predict the current context and that are later sent to the prediction model.

The control buttons are used to control the prediction process. The first button with the name *Recreate model* is used to immediately trigger the prediction process. The second button named *Delete saved data* is used when the user wants to remove the collected data, for example, if the model does not work well. The third button *Send collected data* is used to send the currently collected data for further examination. This button will only be available in a test version of the application.


3:53 LTE

Settings

On device sensors

Collected sensor data

Clicking on this causes recreation of the model and triggers new song prediction

Recreate model

Clicking on this deletes all collected sensor data used for song prediction

Delete saved data

Clicking on this sends all data collected for song prediction

Send collected data

10:51 LTE

Collected sensor values

Time: No
Current Activity: UNKNOWN
Ambient light: 0.0
Is device lying: No
Are cable headphones connected: No
Are bluetooth headphones connected: No
Pressure: 1013.25
Temperature: 0.0
Hashed WiFi name: 0
Connection type: TRANSPORT_CELLULAR
Battery status: CHARGING
Type of charger: AC
Proximity: 0.0
Humidity: 0.0
Heart Beat: 0.0
Heart Rate: 0.0
Location cluster: 0

Figure 4.6: The picture shows the Context Player settings screens. On the left screen, the action buttons are used in connection with the model prediction process. The screen on the right shows the current values that the application has collected.

Notification is another key element of Context Player application. It is a user interface component that must also follow the material design guidelines. These rules determine which elements are to be displayed and how, and also describe several commonly used types that can be applied programmatically through templates.

Context Player has two types of notifications. Both notifications are shown in Figure 4.7. The first is a media type notification, which allows the user to control the media directly from the notification. The media type also allows attaching a specific token generated by the media session to the notification, allowing you to synchronize the playback status of the media service and the current metadata of the media being played. The notification itself can handle up to three actions in a collapsed state or up to five in the expanded view. This allows to add buttons for media control, such as play/pause, skip to next, or skip to the previous media item. Action buttons are displayed in the form of preset icons, which is specific to this type of notification. The media type also enables to register actions as *Media Buttons*, which allows using *buildMediaButtonPendingIntent*, which can send broadcast events to media events that the media service providing songs listens on.

A similar mechanism is used when the application is controlled via Android Auto, headphones with media buttons, or other wearable devices. Along with that, the media notification displays the title, text, and picture that is currently used to display the media metadata. The notification used in Context Player displays metadata in the form of the song title, author, and album art of the currently playing song. It also supports three playback control buttons. These are the play/pause, skip to the previous and skip to the next buttons. After clicking on the notification, another action will be launched, which will open the application on the playlist screen. This notification shows up when the song starts playing and appears during song playback. When a song is playing, it cannot be closed, unless the song is paused.

The second notification that is shown with Context Player is a standard type notification for displaying song predictions. This notification is used to ask the user if he wants to play the song that the application predicted. This type of application displays action buttons with entered text instead of the icon used in the media notification. There are two buttons, *Play* and *Skip for now*, which allow the user to play the predicted song or skip to the next one. The notification informs about which song is predicted to play by displaying its title and author. Performing any action with the notification, such as pressing the play button will start the desired action and dismiss the notification.

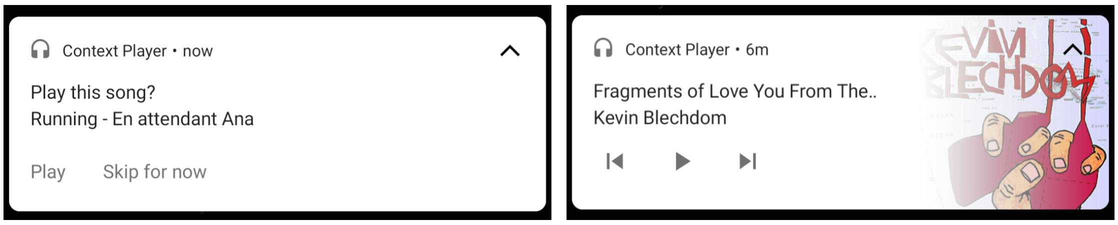


Figure 4.7: The figure shows two types of notifications that Context Player supports. The notification on the left is a prediction notification showing the currently predicted song, and the notification on the right shows a media playback notification.

4.4 Machine learning model specification

This section introduces the machine learning method that will be used in the application. The first part describes the data that will be entered into the machine learning model. The second part presents and specifies the machine learning model to be used in the application, and the last part proposes a solution for the ideal location where the model should be.

Input dataset specification

The data will be obtained from the measured values of the device sensors and from the determined current system context of the application. a more detailed overview of obtaining context data is described in Section 2.1. Data from the sensor will be obtained as follows. First, it is important to check for the presence of physical sensors in the device. The sensor class described in Section 2.3 will then be used to request values from the sensors. The virtual context data will be obtained through the available variables and class methods mentioned in Section 2.3.

The data will be stored locally on the device in a CSV file. This will provide a cluster of data that will be used later in the training of the model. If the size of the stored data allows it to be stored locally on the device, it is likely to be stored in the SQLite[1] database. Saving data locally will also be useful for offline application mode.

The data, with the user's permission, could be used to train a universal model that can later be used as the default model provided to new users of the application. The trained model should then be exported to cloud storage for later inclusion in future application installations.

[1]https://www.sqlite.org/index.html

Application of the machine learning model

An appropriate machine learning algorithm must be implemented to ensure context-dependent behavior. When designing the music player application, a random forest algorithm was chosen, described in more detail in Section 3.1. Random forest is a supervised learning algorithm, which means that it requires marked inputs for training. The inputs are marked by the class to which they belong. The random forest supports a large number of features of the input dataset. The input will be data from sensors and other contextual information obtained from the device. Each entry will be labeled with the selected song or list from which the song was selected. The output will be in the form of a classified song. This means that it is desirable to use supervised learning.

Model's physical location

It is desirable to have a machine learning model physically located on the device. The advantages of this approach are described in Section 3.2. The main benefits include better privacy and the ability to use the application offline without losing any functionality. However, there is a possibility that this option will not be appropriate due to the size, processing capacity and energy consumption of the device. The second option will be to place the model on a server, which will solve the previously mentioned disadvantages. However, the disadvantage of this option may be the potential need to pay for this service.

Chapter 5

Application and model implementation

This chapter describes the architecture and individual implemented elements of the Context Player application. It includes a detailed description of the individual parts of the application (modules), raises the issue of safety and frequency of song playback predictions. Section 5.1 describes the target devices on which the application can be installed. In Section 5.2 is description of the UI elements implementation designed in Section 4.3. Section 5.3 describes the individual application modules. In Section 5.4 there is an overview of the media player component structure, which is one of the two main elements of the application. The last three sections describe the model input data collection and processing in Section 5.5, model construction in Section 5.6 and the prediction process in Section 5.7.

5.1 Target devices

The application was developed to run on devices supporting the Android operating system from API level 21 — that is, from Android version 5.0 called Lollipop. The application itself contains many methods that have been deprecated at a certain API level, or methods that have been recently added and that only support higher API levels. The application was created with a sense to support all functionality across all supported versions. In the case of the previously mentioned obsolete or new methods, there are always implemented methods to create the same overall functionality for all versions of Android from version 5.0 to Android 11 (API level 30).

For fully supported prediction module functionality, the target device should have the following hardware features. The prediction module depends on hardware sensors and components. It usually uses GPS to estimate position, an accelerometer to detect when the device is lying on a table, and a light sensor to detect ambient light around the device. These have been considered essential components that occur in most Android devices today. On the other hand, the application offers the use of many other non-standard sensors, such as humidity or heart rate sensors, for context detection. It is not mandatory to have these physical sensors in the device. The presence of these sensors can be checked using the application settings, where the first list provides a catalogue of all physical sensors available on the device and the second a list of current sensor values that are used for context prediction.

5.2 User interface implementation

The user interface is implemented using the new Jetpack Compose library from Android. Jetpack Compose is a new Android library that enables building UI as components that are part of a code, which is a completely different way than the old approach based on XML resource files. Thanks to that the coupling will be lowered and the cohesion will be higher. This is because in the original approach there is a component called *ViewModel* which provides data to the layout. ViewModel is Java or Kotlin class and is tightly connected with the layout that is written in XML language. This can result in the occurrence of implicit dependencies, about which one part cannot be aware of. The new approach enables writing both parts of this process in Kotlin which results in an ability to make some implicit dependencies explicit, refactor the code and reorganize it which can result in lower coupling and higher cohesion. There is always a UI related logic but now it can be separated more easily. Another reason for developing a new library for UI was the need of implementing complicated UI elements more often, such as animations. This was done in XML file in a more complicated way.

The core component of the Compose library is a *Composable* function. They serve to separate concerns more easily and to make more familiar work, which means working with functions, when creating UI layouts. The structure of it is following. A Composable function is marked with the *@Composable* annotation. Inside the Composable function are invocations of other Composable functions, which represent individual UI components. Along with these functions there can be any other Kotlin primitives, such as *if statements* or *for loops*, to work with the component dynamically. This results in better dealing with the complicated UI logic. This function can be written declaratively, when it is possible to write a description of the desired UI without telling how to get to it, which is making the code clearer and easier to write. It also discards the need of taking care of how to transfer from one state to another, now we only need to specify what the states should be and the transition is taken care of in the framework. The functional composition has an advantage because the subclasses can only have one parent, but functions can be polynomial.

The two items that are mentioned in this section are the playlist and the progress bar identifying the current progress of the song being played. The playlist has been implemented using the *LazyColumn* Compose element, which allows you to draw only the number of items that fit on the screen and lazily redraw them as you scroll, which consumes less memory and provides a faster response.

The progress bar was implemented using the Compose element *Slider*, which is implemented as follows. The slider variable listens to the state of the music player. When the status is *playing*, it constantly reads the metadata value of the current track position provided by the Android *PlaybackStateCompat* class function. The total duration of the scroll bar is retrieved by the implemented *getDuration()* extension of the *MediaMetadataCompat* class. Similar extension functions have been implemented to get song title, author, and album art.

Navigation between screens

The Android Navigation API is used to navigate between the individual screens of the application. The navigation is done by creating the *nav_grah.xml* file that contains representation of each screen included in the application and created connections between them. The visualisation of such file can be seen in Figure 5.1. The navigation itself is initiated via referencing the *nav_grah.xml* in the XML file of the root Activity, which is in this case *MainActivity* class.

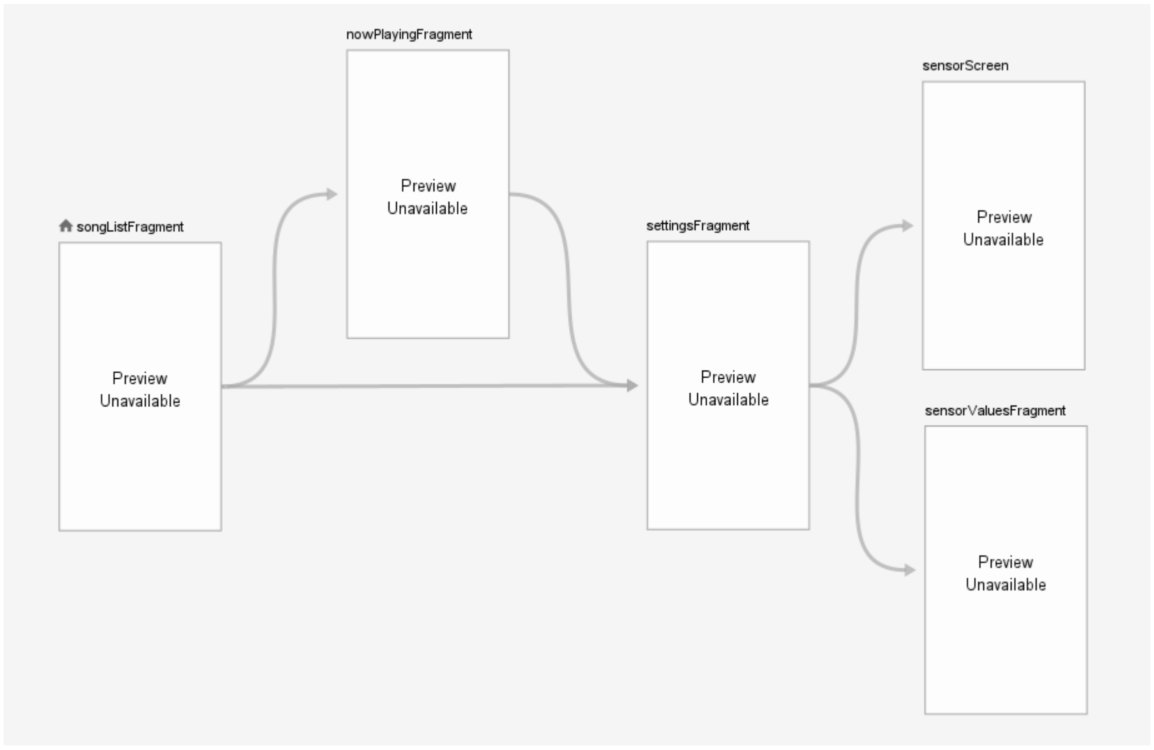


Figure 5.1: Navigation graph determining the navigation between individual screens.

5.3 Application modules

The structure of an application is split into three main modules and one common utility module. The first module is called *app*, which is a core part of the application and includes the application UI, music player with music service and music client and displays notifications. The second module name is the *predictionmodule*, which includes a machine learning model which is used to predict songs. It communicates with the third module called *sensormodule* which deals with sensors. It takes care of getting a permission to access the sensors, reading sensor values and processing them and saving them to CSV file to be later used in the prediction. The last module called *common* serves as a utility module with implemented extension methods and classes which are applicable in all earlier mentioned modules. The extension functions serve to the general purpose common to all implemented modules. Naming convection of modules is according to Android standards, which is lowercase with no special symbols.[1]

App module

This module is a core part of the application. It contains the *PermissionsActivity* which is the activity that is first run at the start of the application. It also includes all UI elements

[1]`https://source.android.com/devices/architecture/hidl/code-style`

of the app and other resources, such as screens, icons, themes, typography. It interacts with all other modules — prediction module, sensor module and common module.

The main component of this module is a music player, its functionality is described in more detail in Section 5.4. The music player is composed of two parts — the music service and the client. The service takes care of a connection to the player which deals with audio files and communicates the player states and song metadata via callbacks to the client. The client is part of the user interface along with a controller that sends playback control commands to the service.

This module also covers a creation and display of notifications. It has a notification manager which cooperates with the music playback service and the function to create multimedia and standard type of notification. Closer descriptions of notifications are in User Interface part of the Section 4.3.

In the module, there is also found UI of the application. The UI is created with the Android Compose library, which is described in more detail in Section 4.3. This module also includes the navigation graph, describing the navigation between screens.

Sensor module

The sensor module is designated to deal with sensor data. The core component is a binding service which, after binding a component, registers the listener with various sensors and listens for changes in their values. It also detects some contextual information such as the type of internet connection, detecting if the headphones are plugged in, etc. The data retrieved from sensors and context listeners are later saved into a custom sensor data class.

Prediction module

The prediction module contains the machine learning model and provides the opportunity of making predictions with it. ML model is created with the Weka library and enables on-device classifications. The input is retrieved from the sensor module in a form of sensor data class instance filled with retrieved data. The module then contains a set of converting functions to be able to convert the data to representations suitable for ML model input and to convert them into arff file which is a more suitable format for Weka models. After prediction, the output in the form of the hashed name of the predicted song title and author is send to the prediction method callers.

Common module

This module serves as a utility module. It contains extension functions to establish server connection and bind to it in one method call, which is a common action in all above-mentioned modules. It also has an abstract class, which serves for generalizing the class used in service binding.

5.4 Media player

The media player is a core part of the application. It serves to list, play songs and it is a framework between user and application to be able to make predictions based on the previous work with it. In the following parts is described the main structure of the player, functionality that it provides, options how it can be used with possible extensions and the main flow of actions.

Media player structure

The structure of the basic media player implemented with native android libraries is following. Each media player is separated into two main parts — player and UI. The player takes digital media and transforms them into audio or video that is ready to be played to the user. The UI part of the media player has two basic functionalities — it receives the transport controls from the user to control the playback and displays the current state of the player with media metadata.

In the case of player part implementation, there are two options that Android provides — a Media Player class which provides the basic functionality of a simple media player, which supports most common audio and video formats and data sources, or ExoPlayer, which is an open source library provided by Android that provides slightly modified functions used to construct the Player part of the application. There are also provided example implementations in form of two demo applications where one is called UAMP and provides display of bare-bone media player application with no dependence on the Player used and the second one is ExoPlayer demo app that shows the advanced capabilities of the ExoPlayer library. ExoPlayer is also not part of the Android framework thus it is distributed separately from Android SDK. In the Context Player is as Player component used Media Player, because it provides basic functionality that is needed from the media player and it is easier to understand the inside structure of the player, which makes it more customizable. In the case of using predictions for song playback, it is also important to have the ability to control it from various components and to modify the playback state accordingly. The overall scheme can be seen in Figure 5.2.

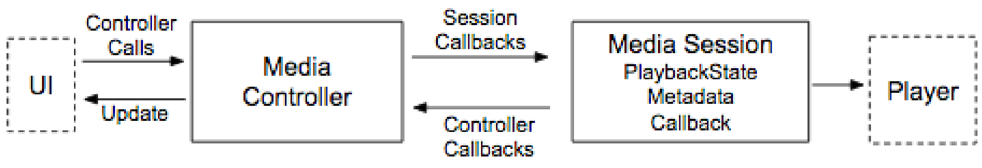


Figure 5.2: Media controller and Media Session scheme (Source: [15]).

Although previously mentioned components, player and UI can be arbitrary, the interaction between those two parts stays basically the same for all media playback applications. Android framework includes two main classes, *media session* and *media controller*, which represent a fundamental structure, that is used for building a media player application. The classes *media controller* and *media session* communicate with each other via predefined callbacks that represent general playback actions such as action play, pause, skip to next, etc. The framework also provides an option to create a custom action which would handle a different functionality aside from the basic one.

The media session is located on the player side and it ensures all the communication with the player from all components. It serves to hide the player's API from the rest of the application and all the calls to the player are made via media session which is controlling it. The session holds information about the current playback state, for example playing state, paused, etc., and about media metadata, such as what song is playing, the author or the album picture of a played song. This provides the possibility to control the playback from the application UI as well as from different devices such as devices supporting Android Auto, Wear OS or devices with media control buttons, for example, headphones. The response to the media session callback is uniform, so it does not matter which client is communicating with the session, it always results in the same action.

The media controller works similarly, but it hides application UI from the player. This results in UI communicating only with the media controller and controller communicating with the player. Media controller serves to translate media control actions that the UI created into callbacks that can media session listen to. From the other side, the controller receives callbacks from a media session when the playback state of the player changes. This provides a tool for automating the UI updates according to the playback change. There is a constraint regarding the number of connections restricting the media controller to allowing it to connect only to one media session at a particular time. This type of structure allows deploying different user interfaces or players at runtime, which can for example ensure better performance or display different UI depending on the device it is running on.

There is a different design in the case of media applications. Video applications usually require to the application be the one that is currently opened in the foreground, compared to that audio application which can run in the foreground or background while the user is interacting with other applications. This results in a different design in both types of media applications. Audio application is specific with the fact that it usually does not have to be visible to be controlled and used by the user. When the application is opened and the song is played, the playback can run as the background task while the user can interact with other applications. This functionality can be implemented in Android by using two basic components — Android activity for the UI and the service for the player. This ensures the desired functionality, because the player will be running in the background thanks to the service component, in contrast to the UI, which can be destroyed after the application is in the background.

In order to implement this server/client logic, the Android library provides two helper classes — *MediaBrowserService* and *MediaBrowser*. The service component is implemented in *MediaPlaybackService* class by extending the MediaBrowserService and contains media session and player [31]. The implemented activity with the UI contains the media controller and the MediaBrowser, which is communicating with the MediaBrowserService. MediaBrowserService simplifies access for the other devices like Android Auto or smart watch finding and connecting to the audio player application, browsing the content and controlling the playback, all without the need of communicating with UI activity at all. There is also a possibility to multiple media apps to be connected to one MediaBrowserService at a time with different media controllers. Because of that, applications using MediaBrowserService should be able to handle multiple simultaneous connections to the service.

Application playing audio should cooperate with other audio playing applications. It should be ready to enable the other applications to play audio. This type of application should also take care of hardware control events from other devices such as headphones. Both parts are taken care of in the Context Player application.

The user should be able to handle audio application volume either by controlling it by the physical buttons on the device or by moving the sliders to change the application volume. The application should also be able to detect sudden disconnection of the headphones to avoid continuing playing music at loud. The volume of the music application should be able to change even if the player is paused between two songs or the song is paused. Android uses different audio streams to control music, system sounds, notifications, incoming sounds, alarms, in-call volume, system sounds and dial pad volume. This enables users to control the volume of each stream individually. The default setting is that the pressing of the volume button will change the volume of the active audio stream. In the case of an audio application, if there is currently no music played, pressing the volume buttons will change the volume of music that would be played or in earlier versions of Android it will change the ringer volume. To set the correct audio stream to be adjusted by volume control buttons, the application should call the *setVolumeControlStream(AudioManager.STREAM_MUSIC)* method. This method should be registered when the application runs, typically from *onResume()* method. Calling of this method results in connecting the volume button actions with the *STREAM_MUSIC* stream, which will control music volume even if the activity or fragment of the player is not visible. There is also an option to control the volume programmatically. This can be handy in case that the desired behaviour of the application would be for example when the user plugs in the headphones, the volume of the music goes to 50 %. However, it is not a recommended approach because it sets the volume of the whole stream for possibly all audio streaming applications. There is also an issue with some devices because not all devices allow the application to control their volume programmatically. This can be checked via *isVolumeFixed()* method call and can be later solved with the call of *setVolume()* method from a particular player, for example from *MediaPlayer*.

In playing audio there is also important to consider the situation that the audio is listened to via headphones, which can be accidentally plugged out. In this case, if the music was played out loud in headphones, after disconnecting them the Android automatically reroutes the audio stream to the on-device speaker and the application becomes „noisy“. The expected behaviour in such situation with onscreen player controls is to pause the playback. In contrast, it is not the expected behaviour for example in games, thus it is not default behaviour. To be able to manage this situation, the system sends the *ACTION_AUDIO_BECOMING_NOISY* intent after the audio output is rerouted back to built-in speaker. With this, it is possible to create the BroadcastReceiver and to listen to this broadcast and in the *onReceive()* method of the BroadcastReceiver, check for the desired intent and pause the playback. The receiver should be registered when the application begins playback and unregistered after it ends it. In the audio application with media session it would be registered in the *onPlay()* method and unregistered in the *onStop()* method. The scheme of the architecture and communication between the client and MediaBrowserService is visualised in Figure 5.3.

It is preferred to use *MediaSessionCompat* and *MediaControllerCompat* versions of the class to provide back compatibility back to the API level 9.

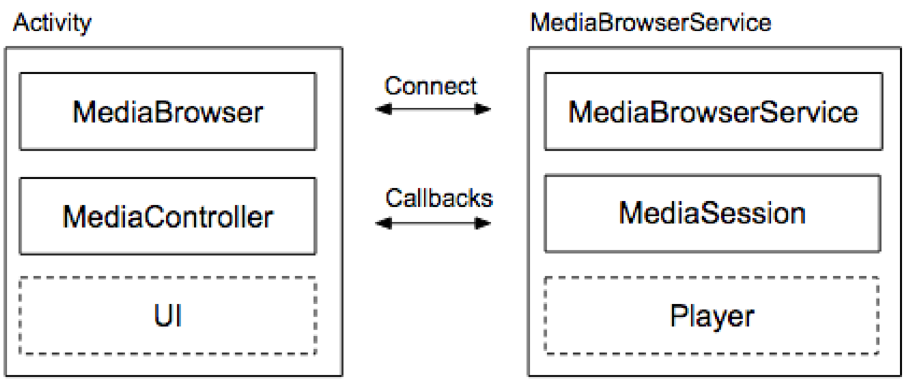


Figure 5.3: The architecture of the UI client and the MediaBrowserService (Source: [15]).

5.5 Context data collection

This section deals with the description of getting contextual information from the device. The contextual information can include individual sensors readings, computed values from obtained sensors, detection of actions performed by the user or events that happen to the device such as that the network connection changed. All of the mentioned entities produce values that can be measured or actions that can be monitored. The following section describes all sensors and events that are used in the Context Player application and divides them into mentioned groups. The second section deals with requesting a permissions to be able to use some of the sensors.

Included sensors and values

There are three main categories for data that are collected in the Context Player application. The first category is physical sensor readings. These sensors are physically located in the device and can vary from device to device. The second group includes facts deducted by computation from physical sensor measurements. The third group is focused on events that are happening when interacting with the device, such as plugging in the headphones. All these types of data are collected and processed to be used together as model input.

The physical sensor readings are performed via listeners, to which has to be application registered. The data are then measured from those sensors, that are physically present on the device, for the sensors not present the zero value is saved. The physical sensors that are considered are following:

- ambient light sensors - detecting the light intensity around the device
- pressure sensor - detecting barometric pressure put on the device
- ambient temperature sensor
- proximity sensor - detecting if there is any phone cover on the device or if the device is close to something
- humidity sensor
- heart beat sensor
- heart rate sensor

The second group are the values derived from sensor readings. This group includes values directly queried from device information. This group includes:

- device is lying detection with the use of accelerometer
- current time
- current location

The third group includes contextual information that are the result of some action performed with the devices. The detected types of events are mentioned in the following bullet list:

- detection of the current user activity — running, walking, riding a bike, traveling by a car or standing still

- detection if the cable or Bluetooth headphones are plugged in
- detection if the device is charging or not
- detection of which type of charger is used — USB, AC or wireless
- detection of the connection type established, if any — cellular, WiFi or Ethernet
- detection of currently connected WiFi name (saved in the hashed form)
- location done by GPS or network provider

All these events are converted to corresponding representation to be able to be used in random forest model and saved to CSV file. The representation of the data is following. The binary data such as if the headphones are plugged in or not are represented as float value 1.0f for true and 0.0f for false. This is for assurance of the right functionality of the prediction model, in detail described in Section 5.6. The events with multiple possible outputs are represented as strings with a short distinctive description of the event. This is due to the acceptance of string nominal values in the random forest model.

The data measurement can occur in two different types — periodical listeners to the sensor value change and on time reading in case of events and device state. In the case of derived values, they were mostly saved in this stage by their raw values. The only value that was converted straight away was detection if the device is lying. The conversion was done with the inspiration of the Hoan Nguyen answer on Stack Overflow[2]. This approach uses the converted accelerometer values to obtain the information about the current device state. The algorithm results in a yes (represented by 1.0) or no (represented by 0.0) answer which is saved to be later in the model.

Another specific case of sensor data acquiring was the detection of the current activity. This was implemented with the inspiration of the explanation article done by Younes Charfaoui[3]. The activity detection was done via the Activity Transition API developed by Google. The individual activities are acquired by registering to the Activity BroadcastReceiver, to which is also supplied a list of currently supported activities and with each activity if the entering or exiting event should be detected. The detected activity is then saved in the same String form which represents the activity in the API. This form is also later used in model prediction.

After acquiring all measured values, they are saved into *SensorData* data class. From here can be accessible when the current sensor values are needed. The values saved here have all default neutral values which can be used in the case that we cannot acquire the value, for example when the sensor is not present or we don't have the relevant permission. The values in the SensorData class have various types of possible values — Float, Double, UInt but also String or more complicated Date.

When the data are requested to be used in model building and performing a prediction, they are converted via *InputProcessHelper* class. The first main method of the class is *inputProcessHelper().* This method conveys the right format of data that has to be represented in a cyclic way. The first information being converted is the date. From date is extracted day of the week as a first variable. The day of the week is in *Date* class represented by the day index, starting from Monday represented by index 1 and ending with

[2] `https://stackoverflow.com/questions/11175599/how-to-measure-the-tilt-of-the-phone-in-xy-plane-using-accelerometer-in-android/15149421#15149421`

[3] `https://heartbeat.fritz.ai/detect-users-activity-in-android-using-activity-transition-api-f718c844efb2`

Sunday with index 7. For the representation used in other machine learning models these numbers were shifted by one index to the left. That means, that Monday has a new index of 0 and Sunday 6. Then it was mandatory to obtain a circular representation of that data. That is because it is important to keep circular nature between the days in the week which means to keep the same distances between all days especially between Monday and Sunday which are represented by the lowest and highest number. This has been achieved via the calculation of a angular distance. The naïve approach would be to represent the days of the week as the points on the circle. This results in representing each day with the 51,5°, which in case if the Monday was represented by 51,5°, then the distance between Monday and Sunday would be 308,5°, which is not equal 51,5° [35]. The right approach is to calculate the sin and cos of these angles. It is crucial to compute both sin and cos because only one representation would end with the same results for different angles, but when put together it results in unique values for all angles.

The equations are following:

$$dayOfWeekSin = sin(dayOfWeek * (2\pi/7)) \tag{5.1}$$

$$dayOfWeekCos = cos(dayOfWeek * (2\pi/7)) \tag{5.2}$$

where angle is calculated in radians. The angle is calculated by the fraction obtained by dividing the full circle (2π) into 7 parts and multiplying each part by the current day of the week index. Both result decimal values are then saved to the CSV file and used in later model prediction.

The second variable that is converted is time. With time it is also desired to have the same circular representation within the day. This requirement results in the use of the same formula but with the replaced representation of a number of fractions which in case of the day of the week is represented by a number of 7. For this purpose, it is needed to get the fraction of the day represented by measured time. This can be obtained via calculating the current time represented in seconds, which can be obtained via the following equation:

$$currentTime.hours * 60 + currentTime.minutes) * 60 + currentTime.seconds \tag{5.3}$$

and then together with total number of second in the day the formula is following:

$$sinTime = sin(timeInSeconds * (2\pi/secondsInDay)) \tag{5.4}$$

$$cosTime = cos(timeInSeconds * (2\pi/secondsInDay)) \tag{5.5}$$

where the full circle is divided by the total number of seconds in the day to get the individual fragments and then to obtain the current number of fragments it is multiplied by the currently measured time in seconds. Both the sin and cos are then saved into the CSV file and later used in this representation as a prediction model input.

The last converted values were the longitude and latitude data representing the location. The problem with this representation is that it represents a 3-dimensional point with two variables, which can lead to weak relations with a point that is in fact strongly related (close to each other). The solution for this situation is to convert the longitude and latitude to X, Y, Z coordinates [9]. The formulas to do so are mentioned in the equations below:

$$X = cos(lat) \cdot cos(long) \tag{5.6}$$

$$Y = cos(lat) \cdot sin(long) \tag{5.7}$$

$$Z = sin(lat) \tag{5.8}$$

These values provide an accurate representation of the device's current location on the surface of the sphere. All these values are then saved to converted CSV file and later used in prediction.

The second method of the *InputProcessHelper* class is called *processInputCSV().* This method converts the initial CSV file called *data.csv* into the *convertedData.csv* with the following changes to the structure. After the CSV file is loaded, the blank convertedData.csv file is created. If the file already exists, it will be overwritten. At first, is to the new file written a header. The header contains all column names that will be in the converted file and later used in prediction. The column names include the class name on the first position, then the day of the week and time circular representation in form of sin and cos values and then the rest of the values that do not need to be specially calculated and are rewritten from the CSV in the same form. Within this method are also extracted all possible values of resulting classes and WiFi names. Then the process of the conversion itself is following. At first, the file is read via the *csvReader()* method. This method with the call of *open()* method opens the inputStreamReader and the call of *readAllAsSequence()* function results in reading CSV file line by line as a *Sequence* type class. Then the map function is used on each sequence, which takes the second item in the row, which is the time and date of the input row. The date needs to be parsed with the according format, which is the E MMM dd HH:mm:ss ZZZZ yyyy„ format that represents for example the date of "Thu Jun 15 10:04:25 GMT+02:00 2021„. The date is parsed with the *SimpleDateFormat()* method. Then in this part is also extracted the ID of the song class that was saved on the first index of the row and added to the list of possible output classes, if it was not already added. Then also in the *map()* method call, is retrieved the number of items in the row and compared to the saved information about the size of the SensorData structure, which was obtained by "csv„ key from the shared preferences file. This information tells if the structure of SensorData data class changed, thus if the data saved in the input CSV file does not match to the current SensorData structure, which would end in error, and thus if it is true, the input file is deleted. This action will not happen often in the future, because it would require change in number of input data saved to the file, which is not usual action. The next action performed in this section is creating the *Location* class with the latitude and longitude variables from the input CSV and adding this class to the list of Location classes. The last activity in the *map()* method call is saving the input data to *Pair* type with the class as the first item and the SensorData structure as the second. When saving the data to SensorData structure, there are all converted from their string representation to which they were read from input CSV file to their according representation they have in the SensorData class, that means the date is saved as the parsed Date class and other values are converted according to their declaration of the SensorData class. After the first section, the second *map()* function is called on the resulting Pair. The first element of the pair is left intact and on the second element, the SensorData data class, is called the earlier described method *inputProcessHelper()*, which converts the date and time data to their circular representation. Then after the data are converted returned by the *map()* method, the *forEach()* method, which is the last method in this chain, is called on the data. The *forEach()* method is called instead of *map()* because the map() returns a processed data that were the input to the method while forEach() does not return a value. This part takes the data from the *ConvertedData* class and saves them to the CSV *convertedData.csv* file. This is done by iterating through the ConvertedData properties, which is enabled by *ConvertedData::class.primaryConstructor?.parameters!!* call. The reason for this call instead of *memberProperties()* method mentioned later is that the order of the items has to keep the same. Each property is then matched with the input data in ConvertedData class and they are saved in proper comma separated format to the new *convertedData* CSV file. In this part, the WiFi hashed names are also retrieved and their individual representatives saved to the list of hashed WiFi names, which is used later in the prediction process.

The last part of the *processInputCSV()* method is the location clustering. This part is working but it is commented out for the case of further development. The main issue is that the clustering algorithm implemented by Gadzhi[4] saves the harvesine distances needed for computation to the matrix that is allocated with the number of input points. In case of Context Player, where as input there are hundreds of points, the application crashes with an allocation error. In Section 6.10, there is a discussion with a possible solution proposed for this problem.

Permissions

To acquire location and current activity data, certain permissions are required. The location permission is needed for a purpose of protecting the user's privacy. The permission is required as runtime permission, which means, that the permission will be requested on the application runtime. In case of location permission, there are many types of permissions to be granted. The type of permission and when to be requested depends of the particular use case. There are basically two types of location access to which the permission is bounded — foreground and background. The foreground access is in case that the application requires location data only for a specified amount of time or only once, for example, one time location sharing in messaging application or navigation to the given location on maps. By the system, the location access is considered as foreground if one of the following situations occurs — if the activity belonging to the application is visible or when the application is running foreground service indicated by a persistent notification. Then when the application is put in the background by opening other applications or turning off the display, the location access stays granted. To ensure that the application gets location updates even when the user chose while-in-use permission, in manifest needs to be registered foreground service with type *location*. On the other hand, the background sharing is for constant location sharing for example sharing of a location between the family members. The system considers accessing location to be background if any conditions for earlier mentioned foreground service are not met. In manifest also need to be specified location permissions - exactly *ACCESS_COARSE_LOCATION* and *ACCESS_FINE_LOCATION*. Before the version of Android 10, the permissions to use location were only needed to be defined in the manifest file, but from Android version 10, they have to be requested at runtime. When the permission is acquired on the runtime it also has to be checked if it was granted when the application tries to get the location data. The permission can be checked via *ContextCompat.checkSelfPermission()* method which returns *PERMISSION_GRANTED* or *PERMISSION_DENIED* depending if the permission was granted or not.

The second event that needs permission to be granted is activity detection. This permission is required to acquire information about the current activity performed, such as walking or traveling by a car. The request is for permission *ACTIVITY_RECOGNITION*

[4]`https://gist.github.com/mgadzhi/b2e8626556ae3b91f9bca1e37268e0ee`

and it is runtime permission, thus it needs to be requested on the application run and checked when the data are acquired same as the location permission.

5.6 Model building

The following section describes the used machine learning model, its construction, description of the tools used and integration into the Android environment. As a machine learning classification algorithm, the random forest was picked. In the first section, the advantages of such a model are described, what its structure and what parameters and settings should be considered when creating it. The second section describes the constraints met and decisions made about deploying the model on a device. The next section describes tried options of machine learning model implementation on the Android platform. The next section presents the final model solution used in the Context Player application. It also describes the data conversion to the suitable format as model input.

Machine learning model algorithm

A random forest was chosen as the machine learning model used for song prediction. A random forest can be a classification or a regression model. In the case of predicting to which group (class) the observation belongs, the classification variant was chosen. The advantages of choosing the random forest are, that it reduces the overfitting of decision trees and thus increases accuracy. Another advantage is that it works well with categorical and continuous data. The advantage is also that it does not require data normalization. The disadvantages include using more computational power, resources and the training takes longer because the algorithm combines multiple decision trees [24]. Another disadvantage is the problem of determining the significance of each variable.

The building block of a random forest is a decision tree. It is based on splitting the input data based on their differing features. The goal is to split the inputs based on their differences that in the result class (group) will be in the same class inputs that are the most similar and the classes will be as different as possible.

The random forest consists of a large number of decision trees. This type of algorithm belongs to a group of Ensemble learning algorithms, which means that connecting more algorithms together results in better performance than if they all worked alone. Each individual decision tree has its own different structure and makes predictions of the input's final class. Then all the predictions are collected and the votes for the classes are counted and the class with the most votes is the result class for the input. The important feature is the low correlation between the trees, which results in better predictions than if the one tree alone decided about the result [45]. This is because when some of the trees can predict the wrong class, others predict the right one, which results in the right class prediction. Thus the low correlation between the trees results in low correlation between the errors. The more trees decide about the result class, the better prediction. To ensure that the trees are not much correlated with each other, the random forest uses two methods — bagging and random feature selection. Bagging, also called bootstrap aggregating, is a machine learning meta-algorithm improving the accuracy and stability of the ensemble algorithms. The random forest is highly sensitive to the structure of training data, a small change in data can result in very different tree structures. The process of bagging in random forest is that it takes advantage from this sensitivity and allows each tree to randomly take samples of decided size with replacement from the dataset which results in very different trees.

Random feature selection is principle based on picking the random feature subset for each tree. In basic decision tree, in moment of splitting the nodes the algorithm picks the from all features the one that produces the largest separation between the observation in the left and right subnodes. In case of random forest, each tree can pick only from random subset of features, which ensures more variability between the trees. The random forest then results in containing trees that are trained on different sets of data and uses different features to make their decisions. This all results in low correlated trees.

The features themselves also need to be suitable for prediction. They need to reflect the predicted class, because if they all would be completely random, the class can't be reliably predicted. The selected features have also high impact on the correlation between the trees.

To construct the random forest model, there are some parameters that need to be taken into account. The first parameter is the number of trees. It is possible to set what number of trees should the random forest have. In general, the more trees are used, the better the result is. But on the other hand the higher number of trees influences the resulting calculation time. The number of trees does not always result in high performance gain, because with higher number of trees there is also a higher computation cost which can outperform the performance of the model. Thus there is a need to find the best ratio between the cost and the performance. The study of optimal number of trees says, that after tested 128 trees in the random forest the other models with 256, 512, 1024, 2048 and 4096 were not significantly better. Also the median and mean AUC (area under the curve) do not change significantly after the 64 trees in the model. This would set the potential threshold for number of trees chosen in the random forest model to be between 64 and 128 trees. The second parameter that can be set is the depth of each decision tree. Generally, it is advised to limit the tree height when dealing with noisy data. The last main parameter is the possibility to set the number of tested features in each node. This parameter is individual and should be tested in order to get the best performance.

Random Forest creation pseudocode, inspired from [10]:

1. Randomly select “k” features from total “m” features where k « m
2. Among the “k” features, calculate the node “d” using the best split point
3. Split the node into daughter nodes using the best split
4. Repeat the 1 to 3 steps until “l” number of nodes has been reached
5. Build forest by repeating steps 1 to 4 for “n” number times to create “n” number of trees.

Random forest on android

In case of deploying machine learning model on the Android platform there are generally three options, possibly four. The first option is to develop an ML model in another programming language and deploy it on a remote server. The prediction is then executed off the device. This choice is connected with the need for an internet connection and results in increased delay caused by communicating and sending data between server and application. The second option is similar to the first one in creating the model in a different language with the difference of converting a model to file that Android can read. The model can be then deployed on the device and the predictions can be run on the device. But there is still a need to send input model data to the server which recreates the model and then downloads the model into the device, which results in the same issue as with the first option. The third option is to use a library or API which is provides an implementation of random forest suitable for Android. This seems to be a best option since the model creation and song prediction could be both executed on the device, thus not needing the internet connection and communication with a remote server which would slow the process down. The downside of this option is that these operations can be slower because other tools for ML can be more optimized. The second and more important obstacle is that the library has to be located in Android. This can be later fixed by adding the custom libraries to the application. The last possible option is to implement whole random forest algorithm from scratch but there is a downside that it may not be optimal enough and there is then a need to also implement all the tools for evaluating and testing a model and for general information about its parameters. For the implementation of ML model in Context Aware application, the third option including tools working with Android was used, which enabled on device predictions without the need for an internet connection and sensor data sharing over the internet.

Used and tested tools for model creation

The initial approach was to create a model outside the application and then in form of converted file use it in the application for on device prediction. The model was created using python in Google Colab environment, which provides to users computing power because there is an option to run the python code on their machines. There are multiple choices of libraries and APIs to be used to create ML model. Often used platform is TensorFlow with its TensorFlow Lite framework which serves for on-device inference and can be used on mobile devices. TensorFlow Lite is lightweight version of TensorFlow to target mobile platforms with constraint capabilities, thus it does not support all operations that TensorFlow does. Because of that, it can be problematic to use an algorithm from the *tensorflow* library and try to use it in TensorFlow Lite framework. The flow of creating the model is following. The model is created using python *tensorflow* library, because the TensorFlow Lite does not provide random forest implementation. The input data has to be loaded to structures and converted to objects called Dataset and process to be accepted by ML model. Then after the model is fitted and tested, it can be exported into file with *.tflite* extension. But there is an issue because TensorFlow random forest implementation includes operations not included in TensorFlow Lite.

Thus it could be converted into *.tflite* file but it would not be recognized by Android and cannot run here. The main reason for the nonsupport of random trees is that TensorFlow is more oriented on neural networks.

The second option of creating and exporting a ML model was to create a random forest model using scikit-learn *sklearn* library and then use custom made converter called *sklearn2pmml* which creates *.jpmml* format files, which can be loaded and used in Android applications. Jpmml format means Java pmml, which means java libraries producing and consuming pmml (predictive model markup language) documents. Pmml documents are files created in XML format which describes data mining and statistical models. After using *sklearn2pmml* converter the random forest representation was successfully saved to *.jpmml* file and loaded into Android application. When the model was loaded in the application, the custom made library called *jpmml-android* from the same author was used to make predictions with the model.

The third option was to use the same random forest model created by *sklearn* library and convert it to *.onnx* file representation. Onnx stands for Open Neural Network Exchange and it is an open file format used to represent machine learning models. To convert the *sklearn* model to *.onnx* representation, the *skl2onnx* python library was used. After converting the model and loading it into Android application, the *onnxruntime* library for Android was used to make predictions.

The next tried option was to convert the random forest model into basic *.pmml* file format and use *pmml4s* library to make predictions with it on Android platform. This was the first working option. After loading model into *Model* class and filling it with mapped inputs, it was able to produce a list of output probabilities for each output class with given output. This option can be used for further testing of model prediction done off-device.

The last option was to create an on-device evaluation with the use of Android libraries or APIs. For this purpose, the *weka-android* library was used. Weka is machine learning open source software written in Java that collects machine learning algorithms for data mining tasks. Weka can be run on Windows, Mac OS or Linux and has a graphic user interface. There is no official android library made by Weka authors, but android developers took the source code and removed GUI parts, which Android Java implementation (Dalvik) does not support to be able to run Weka on Android. The original library that does this is called *Weka-for-Android* from rjmarsan. The library is in *.jar* format. This library was created from an old version of Weka 3, thus in need to support the newer version Matt Schuchhardt created his library *android-ml-weka.* The latest library was created by andrecamara and its name is *weka-android* and it is a repackaging of the previously mentioned library to the android library. It is available in *.aar* format. Weka for Android was used as the final solution in Context Player application.

Weka for Android in Context Player

In the Context Player application was as the ML model for creating song predictions used library *Weka-for-Android* mentioned in the section *Used and tested tools for model creation.* This library was integrated into Android environment via downloading the library file with extension *.jar* from GitHub page. Then the library was added to *libs* folder, where it could be later detected as Java library. Then the library was added to dependencies of the prediction module to be later used for model creation. The model is created in the *PredictionModelBuiltIn* class of the prediction module. The class *RandomForest* from Weka is instantiated here and the model is built with training dataset by using *buildClassifier()* method. The possible parameters of the model are the number of trees which defaults at 10, maximum depth of the trees, option to set a random seed and number of input features. Then the model is evaluated and the current information about the model is displayed. Then there is a function called *predict()* which after calling with input as a parameter calls function *classifyInstance()* on the created model.

To be able to classify the data with weka library, it is required to convert the data to file of ARFF (Atribute-Relation File Format) format. The ARFF file represents a file containing a list of instances with their representation by attributes. This type of file was developed by The University of Waikato to be used with Weka machine learning software. ARFF file contains two main sections — Header and Data. The Header contains the relation name, that is the name of the dataset and the list of attributes (which are derived from the header of input CSV file) and their types of values. There are also possible comments starting with the % sign and the keywords @RELATION, @DATA and @ATTRIBUTE are not case sensitive. The example of the ARFF file can be following:

```
% This is file with converted data
@relation convertedLocData

@attribute class {2046003820,3758008002}
@attribute sinTime numeric
@attribute dayOfWeekSin numeric
@attribute light numeric
@attribute state {IN_VEHICLE,STILL,WALKING,RUNNING,UNKNOWN}
@attribute batteryStatus {NONE,CHARGING,NOT_CHARGING}
@attribute chargingType {NONE,USB,AC,WIRELESS}
@attribute humidity numeric

@data
2046003820,-0.74295,0.433884,224,STILL,CHARGING,AC,0
3758008002,-0.742463,0.433884,227,WALKING,CHARGING,AC,0
3758008002,-0.741976,0.433884,226,WALKING,NOT_CHARGING,NONE,0
```

Figure 5.4: Arff file which is used as an input to Weka model.

The attributes can be of four basic types. The first type is numeric, which can be Double or Int number and is specified with the *numeric* keyword. The second possible type is nominal, which represents categorical values. All the possible variants of the nominal values need to be specified in the curly brackets behind the attribute name. The enumeration of possible values is used instead of the keyword. The third possible type is string type, which can be used in case that the inputs to the machine learning model are for example sentences for case of text analysis. There is no enumeration of all possible strings thus the *string* keyword is used to identify the type. The last type is the date type. There can be together with the *date* keyword specified the date format of the input values. The last special type is nominal type with attribute name class, which can be later used as output values for training and testing of the classification model.

In order to create an ARFF file there are two options, write it manually or use weka tools to transfer files of other versions into ARFF format. The input file format is CSV file with the header with names of columns, which will be later used as attribute names in ARFF file. In Context Player are data saved in raw format to csv file and then processed and the header is assigned to them. After the file is prepared, for data to be loaded from CSV file it is needed to instantiate *CSVLoader* class which loads the data, convert them into *Instances* via *getDataset()* method. To perform the conversion to ARFF file, the class *ArffSaver* needs to be instantiated. Then data needs to be assigned to the ArffSaver, for which are used the created Instances. After that, it is required to specify a name of the new file with the *.arff* extension. Then the *writeBatch()* method is called which results in converting the Instances and writing them into *.arff* file.

After obtaining the initial structure of the file, there are still some imperfections that need to be fixed to ensure that the will model works well. To fix these issues the following approach was taken. The first issue is that the converter does not recognize which attribute is the class attribute, which results in need of naming the column with output values as class and in case of numerical class labeling also with converting the attribute type manually.

Unfortunately, there are no methods to covert attribute type, thus it needs to be done by searching and replacing the types keywords in the ARFF file. Another problem is that even if the attribute is recognized as a nominal type, it still does not include all possible values which the nominal attribute can acquire. Thus the possible values need to be included by replacing the original list of attributes with the new one. But it is necessary to have all the possible nominal values. In case of predefined groups for example when detecting the type of charger connected, there are four possible values — USB, AC, wireless or no charger. But in case that we don't know the number and names of the types beforehand, for example in case of hashed WiFi names of output classes, it is required to list them from the input CSV file.

In case of predicting new values, there is a little help because it is possible to create a new instance with the Weka library methods. There is a possibility to create an attribute with the *Attribute* class, specify the type by the value or by providing a list of possible values together with the input to specify the nominal type. Then the attributes are added to the list from which is created the *Instances* class instance. Then with the method *setClassIndex()* it is possible to specify which is the output attribute. After that, the *DenseInstance* instance of a class is created, in which constructor are supplied input values to the attributes. Then the values are linked with the attribute structure via *setDataset()* method and this serves as an input to the prediction model.

5.7 Prediction process

This section describes the flow of constructing and triggering the prediction. It deals with scheduling a backgrounds tasks that will trigger the prediction. For task scheduling, there are described many techniques and the used one — scheduling with WorkManager. Then in the section *Prediction creation* is described the whole process of creating a prediction.

Scheduling of a prediction

The prediction of a song should be an event conditioned by a specific context. It is, therefore, necessary to determine when and under what circumstances the forecast should be run. The goal is to predict songs in a given context, even when the application is not running.

The first option to consider is to anticipate playing a song using a special event that distinguishes different contexts, such as connecting headphones or changing user activity. The ideal solution would be to use the *BroadcastReceiver* class. In order to listen to such actions, BroadcastReceiver must be registered in the *Manifest* file, the class must be extended, and the *onReceive()* method must be overridden. BroadcastReceiver can listen for these events even when the application is not running or if the device has been rebooted. However, the first problem with this option is that the default BroadcastReceivers, which means that registered in the manifest and independent of the running application, do not run from Android API 27 (Oreo version) because they drain too much battery, which is against Android's new policy. The naïve solution would be to require permission to disable battery optimization for this application. However, this would be disadvantageous for users due to high battery consumption and higher APIs do not allow implicit broadcasts to run.

The solution suggested by the Android developer guidelines and used in Context Player is to use *WorkManager*. WorkManager is an API for scheduling asynchronous background tasks. It supports Android API from level 14 and is optimized for lower battery consumption. This is the recommended option, which replaces all previous background job schedulers, such as *FirebaseJobDispatcher*, *GcmNetworkManager*, or *Job Scheduler*, but still uses them at a low level. Specifically, the Android API level 23, which Context Player supports, uses Job Scheduler. WorkManager was based on time-dependent scheduling tasks, so the prediction scheduling logic had to be changed from activating a specific event to periodic time activation and event occurrence checking.

There are two types of performed *Work* — one time work requests and periodic requests. Periodic applications would be a good choice, but they have the disadvantage that the minimum time between two work requests is 15 minutes. In order to be more flexible with periods, a one time work request must be used. Then, the periodicity of scheduled tasks is ensured by scheduling the first task when the application starts, and at the end of the scheduled task, it is rescheduled with a specified delay, which can now be less than 15 minutes.

For this purpose, it is important to set that the work could have only one unique instance by calling method *enqueueUniqueWork()* with a similar ID for the initial Work and Works scheduled by it. Then the policy in case if there is already an existing work with given ID needs to be specified. There are options such as *KEEP* that ensures that already running work would be kept, *APPEND* which appends second Work to the existing one and *REPLACE* that replaces already running work. This option is essential to set up in case of application start, when in the *MainActivity* class is called the initial WorkManager. The option to be chosen here can be *KEEP* to ensure that the period is still ongoing or *REPLACE* which ensures that every time the application is open there is prediction ready, which is used in Context Player.

WorkManager allows to set up constraints under which tasks can be run. This can appear similar to Broadcast Receiver but the logic is different. The task still runs after a given period of time or instantly, but the given rules can decide whether the work will be performed or not. The requirements that can be set are constraints that the storage can't be low, the battery can't be low, the device is charging, the device is idle or is connected to a specified network type. It may seem that it is still possible to emulate a Broadcast Receiver by including a one-off job with a very short period of time and given the constraints to be met. However, this option results in high battery consumption, which is undesirable for mobile applications.

The work itself is specified by overriding the *doWork()* method in *CoroutineWorker*. Here, *CoroutineWorker* is used instead of the basic *Worker* because it provides a suspendable call to the *doWork()* function.Inside *doWork()* method is created instance of a class *MediaBrowserConnector* which serves to trigger a prediction and display a notification with a predicted song. The instance is created again each time because the application does not have to be running when the scheduled work starts to run, thus the class does not have to exist. Creating instance of the *MediaBrowserConnector* class is also constraint by check if the file with input sensor data exists. If the data was not collected yet and the file does not exist, then the class does not have to be created because the prediction will not run. After this check and probable creating of an instance, the background task creates a new copy of itself and schedules it with a given delay.

For the purpose of monitoring WorkManager offers to add observers to the state of the Work. Work can be in the following states — enqueued, running, canceled and after performing work succeeded or failed. Enqueued state is after calling method *enqueueUniqueWork()* which ends in the task being scheduled with given delay. Then after the given period of time the task change to state running. In enqueued and running state is Work consider uncompleted, thus the policy of keeping or replacing the task applies there. During the entire run time, the task can be canceled via explicit call. The new version of Android Studio Arctic Fox offers a tool for monitoring background tasks called *Background Task Inspector*, but it is only available for applications from Android API level 26. It is a new part of the *App Inspection* utility. It allows developers to real-time watch how many background tasks are in which particular state, the ID of the task, which type of tasks they are (one time or periodic) and in which class is the work for this task specified. Important information in the Background Task Manager is the time when the task was scheduled and by which class it was. In case of Context Player this helps to distinguish between the initial task scheduled from the *MainActivity* and the rest and also helps to monitor whether the new tasks are being scheduled because of changing schedule time.

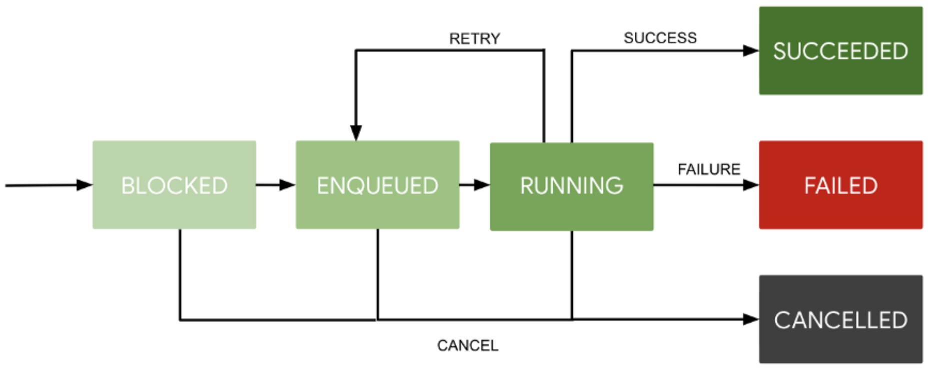


Figure 5.5: Context Player launcher icon (Source: [16])

Prediction creation

The process of creating the prediction is following. At first, the scheduled work initializes the *MediaBrowserConnector*. It connects to the MediaPlaybackService to prepare player. The connector also binds to the MediaPlaybackService to get the song's metadata. This is done by *bindServiceAndWait()* method, which construction is standard but the way of handling it in coroutine was inspired by the answer on Stack Overflow[5]. Then it creates the ML model. After creating the model it reads the current sensor data and triggers a prediction on them. The prediction result is a predicted class hash code which is decoded in *MediaBrowserConnector* to identify which song was predicted. After that, the notification is constructed with the metadata about the currently predicted song.

[5]`https://stackoverflow.com/questions/48381902/wait-for-service-to-be-bound-using-coroutines`

Chapter 6

Model and application testing, result discussion

This chapter provides the overall testing of the application functionality, user interface, evaluation and testing of a prediction module itself, user testing of a application a prediction accuracy. It also includes description of the application publishing, more specifically publishing on the Google Play platform as an music application and publishing on the GitHub as an open source machine learning project.

6.1 Testing of application functionality

The second form of testing was focused on the application functionality. This section discusses tests of the functionality of an application that either worked correctly or encountered errors or unpredictable behavior. The application has been tested with a number of testers and potential users of the Context Player application. The application was tested via Google Play Console platform, which provided an option to test the application on multiple devices.

Internal testing on Google Platform

The user testing was performed via the Google Developer platform. This approach was chosen for a number of reasons. The first reason is the convenience of testing the application on various devices with different users without the need to meet in person. The second reason is that the application was about to be published on Google Play, hence it is advisable to meet all the requirements in advance in the trial version than in the later official version of the app. The downside of using this platform is that when the new application update was released then the testers received it a couple of hours later. The same problem was with reported crashes which were delivered hours after the actual crash happened which slowed down the debugging and testing process. On the other hand, the upside of this solution is that the testers can become future users and when the official application release is published they can just update it and continue to use it.

The testing was conducted in the following way. At first, the testing (debug) version of an application was loaded into the Google Play Console. The same information as in the official release needs to be filled in, such as the target age category, information and specification about the application. Also, the countries in which the testing can be conducted must be chosen. The only difference against the release version is that it does not have to be concerned about GDPR rules. Then, the list of testers was specified with their email addresses. Then when the tester received a link to the download of the testing version, they downloaded it as normal application. When the change in the application was done, the new version of an application was loaded and the testers received the update on a same place as all other applications installed via Google Play.

The testing on the Google Platform provided crash reports that were later used to repair the errors within the application. Each user was instructed to use the application as they wanted, with an explanation of the basic functionality. After the first install, all crashes and Application Not Responding (ARN) reports were received into the Google Play Console.

All the reported crashes were fixed and the application runs on all tested devices, which were from various range of types and Android versions. The tested brands were Xiaomi, Samsung and Sony and the API levels tested were 25 (Android 7.1) and 29 (Android 10).

Results of tests and solutions

Many of the crashes happened on the first install of the application, when the application could not be run. Most of the crashes were due to unchecked conditions and unexpected values of the sensors on other devices.

The first major problem reported was that the application crashed during a new installation because some sensors needed to check if they were present before trying to read from them. The result was an error caused by uninitialized sensors. The error was corrected by checking if a sensor was present.

The other group of problems was caused by unpredictable sensor values, such as accelerometer which not returned the default three values or different behaviour when the WiFi name was not acquired. This was handled by checking for these values and returning the default value if they occurred.

A big issue was also determining the right format of ARFF file which is the input to the prediction model. There were problems mostly with nominal and binary values, where binary values needed to be declared as nominal values and nominal values had problem when they were represented as numbers they needed to be converted to Int and then to String and as a String added to the nominal values list of the particular variable.

The last group of occurred errors were application not responding (ANR) errors. To resolve this problem the run of the application was optimized by running the computations such as model prediction, data collection and conversion on the different thread by calling *withContext(Dispatchers.Default)* which runs the computations on the *Default* thread which is designated for such work.

User feedback

The feedback of the application was collected from all the users. There was a possibility to leave a feedback via the Google Play Console or via email but the majority of the feedback was send through the personal messages. The main comments on the application functionality or design were discussed and later implemented into the application.

The first proposed change to the design was to build an action bottom bar, which would show the currently played song and which would allow to pause and resume the song playback. The bar should also be displayed only when any song was chosen to play. The component was implemented and in Figure 6.1 is the final design within the application.

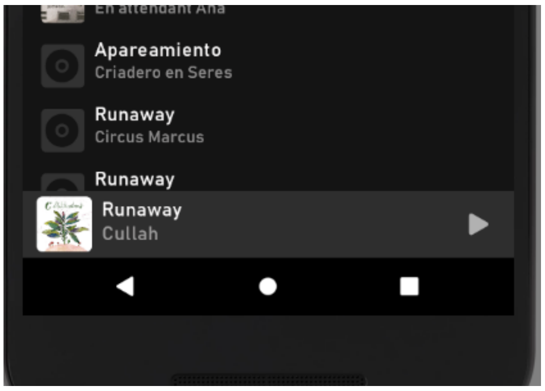


Figure 6.1: Implemented bottom bar player. It provides playback control and displays currently played song.

6.2 Model testing specification, data collection

The model was tested with several users on their devices. The following section describes how the testing was carried out, what aspects were tested, the amount of data collected and the tested results. The first section describes the formulation of the testing task, which means what were the instructions set for the testers and what were the testing conditions and result requirements.

The testing formulation

The testing requirements were following. Data were to be obtained from real measurements to ensure that the measurement result is as close as possible to the real outputs of the application model prediction. The goal was to measure at least four different contexts from each tester. In each context, there should be 1 to 3 songs played repeatedly that in the end there would be about 100 collected data from each situation.

The testing conditions were specified to try to distinguish the contexts with at least one decisive action, such as headphones plugging in, internet connection change, etc. For one model there were also requirements to try to distinguish the different contexts only by changing the physical sensor values, thus not providing any mentioned decisive action.

The testing itself was carried with the four testers. Each tester received instructions in the following form. Each tester downloaded the testing version of the application from the Google Play Store. After each user got instructed on how to work with the application the data were collected by playing songs. The songs were loaded from the local storage of the testers or there were sent links to free downloadable music files. The was positive fact was that three of four testers already have songs locally on the device and could test the application functionality immediately. This fact also confirmed the assumption from the application design, which concluded that many people have their songs locally on the device, thus it is a primary location to load the songs.

Each user was instructed to play different songs with different activities (contexts). There also could be exceptions in song choice and some of them could appear in multiple contexts. After exiting the current activity, each user was asked to send the collected data via an automatically filled email. The only thing that was required to fill in was the subject of the email with the description of the current activity. The number of contexts was not restricted. The contexts also appeared more than once to confirm the prediction functionality. There were also sent reports only with updated predictions.csv files, which data was collected when the user performed the already recorded activity.

The testing version of the application had the following parameters. The data were collected every 10 seconds when the song was playing and the collection began after the initial 10 seconds of the song playback to ensure that the representative context data were collected. The prediction itself was triggered every minute and was not conditioned by any change of context (which is conditioned in the published version). Also, the button to send the testing data was located only in the testing version of the application.

The data collection

The data collected from each tester were collected in the following manner. After leaving the current activity, each tester sent the data via a button created for such purposes. After pressing the send button the prompt for selecting the email application showed to the user. Then after selecting the email application the email was automatically filled with the recipient, subject and the files with collected data attached. This automation makes the process of reporting the different contexts easier.

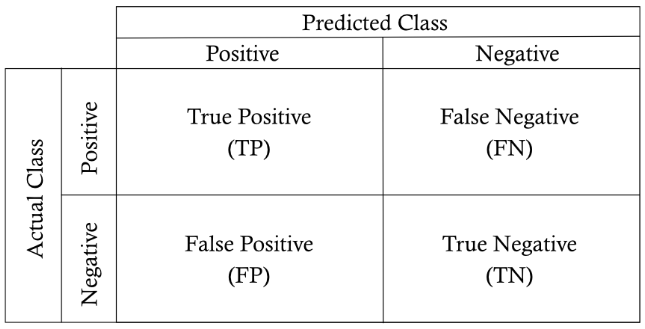
The data were send in form of original collected data, converted ARFF file and CSV file called *predictions.csv*. In the predictions file was stored the class of a currently predicted song and the data collected from the current context based on which the prediction was triggered.

The data were sent multiple times based on change of the activity. This was done this way to keep separated the data from different contexts for testing purposes. On the receiving side, the data were manually stored and organized and the different contexts marked. The description of the current activity was required to be written in the email subject.

The dataset sent in the end was the data collected from all the activities performed with all the predictions together. The total number of contexts was different in each dataset.

6.3 Model evaluation metrics

When evaluating the model, there are several metrics that are usually used in classification model evaluation. When using the metrics, it is important to set up its basic components. These are the division of the output prediction into four basic groups of *True/False Positives/Negatives.* These metrics can be represented via a confusion matrix. The structure of confusion matrix for binary classification (classification with two possible output classes) is shown in Figure 6.2:



		Predicted Class	
		Positive	Negative
Actual Class	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 6.2: Confusion matrix representing the possible model output.

The traditional machine learning model evaluation metrics are following. [3]
The metrics are following:

- *Accuracy* — Accuracy is a typical classification metric. It is easy to understand convenient method for multiclass classification. The downside of this metric is that in case that one class is dominant over the others (for example it is the result of 95 % of the training data) then the accuracy can be very high, because when the model will be evaluated with the same distribution of the data it is enough to always predict the majority class, but the model will not predict well the minority classes [2]. The formula is simply all truly predicted results divided by the total number of all samples classified. The result value is often represented in percent. The formula is written as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{6.1}$$

- *Precision* — precision measures the capability of the model to identify correct instances of each class. This metric answers the question of what is the portion of truly positive outputs from all predicted Positives. This metric is useful in cases when is desired to have very accurate results. It represents the ratio between correctly identified class and the whole dataset. The values are in the range of 0.0 to 1.0. The formula is:

$$Precision = \frac{TP}{TP + FP} \tag{6.2}$$

- *Recall* — this metric retrieves all correctly classified instances per class. It answers the question of what is the portion of correctly classified true Positives of all positively classified instances. The metric is suitable for cases when it is important to have the majority of all positive results of predictions. The formula is following:

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

- *F1 Score* — this metric is the harmonic mean of precision and recall and it is normalized between 0 and 1. F1 score preserves the balance between the classifier's precision and recall. Precision and recall are inversely related in this metric. The value of F1 score 1 indicates the perfect balance between the two. The metric is used in cases when the balance between recall and precision is desired. The formula for calculating the F1 score is following:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6.4}$$

- *OOB error* — the out-of-bag error is used in classification evaluation, to set the classification error. This metric is connected with the bagging of the random forest algorithm and it is estimated internally during the algorithm run. The logic of the algorithm is following. During the bagging phase, each tree construction is based on a different bootstrap sample made from the original dataset. About one-third of the input data is in bootstrap and thus are not used in the particular tree construction. Each of these unused values are then used to evaluate this particular tree classification.

- *Mean squared error* — Mean squared error or MSE is the error metric used in machine learning model evaluation representing the quality of the estimator. It is the square of the difference between the predicted and actual target classes, divided by number of input samples. RMSE is square root of MSE and it is also used as the error metric in machine learning model evaluation. The advantage of the RMSE is that it is measured in the same units as the target variable [1].

Metrics use on multiclass classification

These formulas are representing the case of binary classification, such as if something belongs to the class or not. For the purpose of using those formulas in multiclass classification, there are several principles that can be applied. The difference in the multiclass classification is that there are no positive and negative classes. Thus those classes need to be represented in a different way. The principle is based on determining the TP, TN, FP, FN of the each class.

The demonstration of acquiring the mentioned values is shown in the following explanation. In this example, the values of TP, TN, FP, FN are acquired for one class and in the model are possible three output classes. In Figure 6.3 are represented the predicted classes and the *Song 1* is a class for which the calculation is currently done. The calculation is following. The Truly Positive outputs are values in the top left corner, that means that they were predicted as *Song 1* and were actually *Song 1*. The Truly Negative (TN) are these that were not classified as *Song 1* and also shouldn't. These are the negatively predicted numbers in the second and third row. False Positive ale the values that were predicted as *Song 1* and actually were not this class, thus the values in the first row without the positively predicted values. And the False Negatives are the values in the first row without the positive predictions.

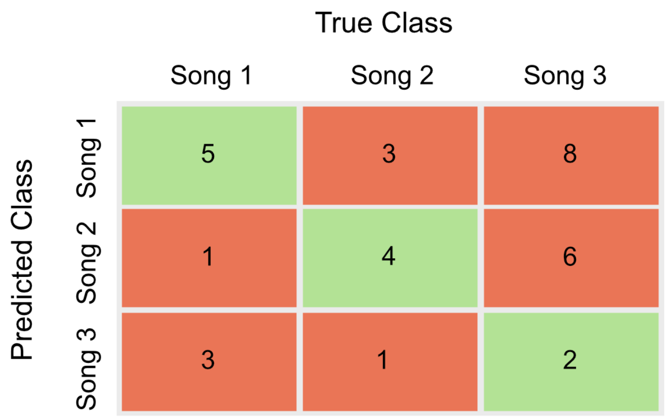


Figure 6.3: Confusion matrix representing the possible model output for three classes (songs).

The result values of the above example are:

$$TP = 5 \tag{6.5}$$
$$TN = (1 + 6 + 3 + 1) = 11 \tag{6.6}$$
$$FP = (3 + 8) = 11 \tag{6.7}$$
$$FN = (1 + 3) = 4 \tag{6.8}$$

From this values it is possible to calculate Precision, Recall and F1-score with the proposed formulas. Then the same can be done for all other classes. Then the overall metric result can be obtained from these values by multiple ways. It is possible to combine the F1-score result in multiple ways to obtain the overall metric for the whole model. This can be done by the following ways:

- *Micro F1* — this variant is called micro-averaged F1. It is computed from total of TP, FP and FN of all classes [37]. This metric considers all classes globally. This metric can be represented that all samples equally contribute to the final average. It is suitable for models with multi-label problem, which mean that the sample can be classified into more than one class. Thus the values from the previous example will be:

$$TP = (5 + 4 + 2) = 11 \tag{6.9}$$
$$FP = (8 + 7 + 4) = 19 \tag{6.10}$$
$$FN = (4 + 4 + 14) = 22 \tag{6.11}$$

  and the calculated metrics will be:

$$Precision = \frac{11}{11 + 19} = 0.37 \tag{6.12}$$

$$Recall = \frac{11}{11 + 22} = 0.33 \tag{6.13}$$

  the F1 result is then obtained by the classic equation:

$$MicroF1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 0.34 \tag{6.14}$$

- *Macro F1* — Macro F1 calculates also all the basic metrics individually for each class as the Micro F1 and then calculated the unweighted mean of all F1-scores [37]. The metric can be characterized by the fact that all classes are contributing equally into the result average. Thus this metric is useful in case that the input dataset is strongly unbalanced. From the previous example are the values following:

$$MacroF1 = \frac{0.40 + 0.42 + 0.19}{3} = 0.34 \tag{6.15}$$

- *Weighted F1* — Weighted F1 is similar metric to Macro F1 but it calculates weighted mean of the F1-score results [37]. This metric can be understood as that the contributions of individual classes into the final average are weighted by their sizes. The weights for each class are obtained from the number of samples in each class, the formula is then following:

$$WeightedF1 = \frac{(0.40 \cdot 9) + (0.42 \cdot 8) + (0.19 \cdot 16)}{9 + 8 + 16} = 0.30 \tag{6.16}$$

6.4 Used model evaluation algorithms

This section describes testing and evaluation algorithms used in Context Player model evaluation. The evaluation was preformed via the methods from Weka library which also delivered calculated metrics from the section 6.3. This section also discusses the preparation of the data to the evaluation algorithm. The algorithms are used to produce the metrics mentioned in the section 6.3.

Preparation of a testing data

The machine learning model was created with the Weka library which also provides tools to evaluate and test the model. In order to create model evaluation and perform testing, the input dataset needs to be separated into training and testing data. This is done in order to keep the testing of the model unbiased. At first the data need to be loaded from the ARFF file with the *ArffLoader* class. After instantiating the class the input file needs to be specified and then data can be loaded with the *getDataset()* method. Then after setting the output class attribute the data are randomly shuffled with the Weka *randomize()* method.

After loading the dataset, the *RemovePercentage* filter class was instantiated in order to split train test data. The filter is used twice, once to get the train and once to get the test data. At first the percentage of splitting the training and testing data needs to be set. In this case was data split to 80 % of train dataset and 20 % of test data. After setting a percentage the filter is applied on on dataset in order to create train set. Then the same filter is used on the dataset again but with option *invertSelection* set to true. This results in taking the complementary 20 % of the data and saving it to test set. The both train and test set are saved to instances of the *Instances* class. After that the training dataset is used to train the model and testing data are used for evaluation after the training.

Model evaluation with testing data

The first method that can be used is called train-test split method. This is the simple evaluation method that uses a portion of the input dataset to train the model and the rest for the testing. The data can be shuffled but it is not required by the algorithm. The downside of this method is that the data are split only once which can result in uneven split, which results in possibility that some class will be present only in one part (in testing or training data only).

The evaluation itself is done by instantiating the *Evaluation class* and calling its *evaluateModel()* method, which inputs are trained model and testing dataset. After the evaluation the Weka offers methods to print out the calculated metrics. Values that output the evaluation algorithm are textual representation of a correctly classified instances and errors percentage, the model confusion matrix which is closely described in section 6.5, then the metrics calculated for each class and the list of every predicted value for every input.

Cross-validation

Cross-validation is model evaluation technique, that presents how well the model reacts to the generalized input. This technique serves to estimate the model performance. This algorithm was used to test the performance of the random forest model and to tune the hyperparameters of the model. There is a number of different types of cross-validation techniques, but they are all based on the same algorithm. The algorithm is following:

1. Split the initial dataset into training and testing data.
2. Train the model on the training set.
3. Validate the model on the test set.
4. Repeat steps 1 to 3 a couple of times. This number depends on the CV method that was chosen.

There are many types of cross-validation algorithm. The type used in Context Player cross-validation is K-folds. K-folds cross-validation has advantage over other type called Hold out, which separates the data only once. The problem with Hold out method is that for example the data in training dataset do not have to be present in testing dataset, thus one phase can be harder then the other. In case of K-folds technique, the testing is performed on the multiple different subsets of the original dataset, which ensures more stable and reliable results. The algorithm of K-folds cross-validation is following:

1. Choose the number of folds — k. Usually, k is 5 or 10 but any number which is less than the dataset's length can be chosen.
2. Split the dataset into k equal (if possible) parts (they are called folds).
3. Choose $k - 1$ folds which will be the training set. The remaining fold will be the test set.
4. Train the model on the training set. On each iteration of cross-validation, a new model must be trained independently of the model trained on the previous iteration.
5. Validate on the test set.
6. Save the result of the validation.
7. Repeat steps 3 to 6 k times. Each time use the remaining fold as the test set. In the end, there should be model validated on every fold present.
8. To get the final score average the results saved in step 6.

The algorithm itself is visualized for ten folds in Figure 6.4.

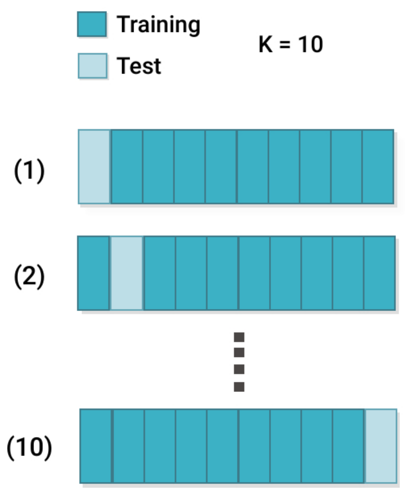


Figure 6.4: Cross-validation algorithm visualised for ten folds (Source: [33])

The number of folds is parameter that can be set, the default number is usually set to ten. The evaluation can be made even more reliable with the higher number of folds [34]. But on the other hand increasing the number of folds results in increased computing cost and time consumption. There also exists an improved version called Stratified k-Fold, which counts with unbalanced dataset, for example when one class is represented much less then other classes.

Cross-validation in Weka environment provides K-folds variant with the adjustable number of folds. Weka also offers to use Stratified version of the K-fold algorithm. This ensures better evaluation for model with nominal values, which is the case of the Context Player model. The algorithm is run with calling *crossValidateModel()* of the *Evaluation* class. The parameters supplied to the method are classifier, in our case random forest, training dataset, number of folds and random seed. There are multiple ways The number of folds was set to ten, because this was verified by experiments that this number produces model skill estimation with low bias and high variance [8]. After running the algorithm, the number of metrics are received, which actual results are mentioned in the section 6.5.

6.5 Context Player model evaluation and testing, results

This section describes metrics used to evaluate the Context Player application model. Together with metrics are described which algorithms were used and what data were acquired from them.

Confusion matrix

The first model evaluation technique that was used was printing of confusion matrix. The confusion matrix is not a metric itself but it is a diagnostic tool which gives the overview of the prediction results. It has form of a table and its cells are used to calculate the actual metrics. a confusion matrix is representation of how many predictions done by a model were correct and when the prediction were incorrect, what was the incorrect prediction. In the confusion matrix produced by Weka library are actual classes in the column and in rows are the predictions made for the input. The numbers on the diagonal represent the number of correctly predicted classes. Every other value than on diagonal represent wrong prediction made by a model. In the column is value that the predictor has chosen and the row represent the actual class. This type of overview is useful in case to detecting the possible areas where the model decides wrong in the similar pattern.

```
=== Confusion Matrix ===

a   b   c   d   e   f   g   h   <-- classified as
6   0   0   0   5   0   0   0 |   a~= 2046003820
0   2   0   0   1   0   0   0 |   b = 3758008002
1   0   0   0   0   0   0   0 |   c = 4027449371
0   0   0   2   0   0   0   0 |   d = 1242139011
2   1   0   0 596   2   0   0 |   e = 2628202871
0   0   0   0   0   3   0   0 |   f = 1591806489
0   0   0   0   0   0   0   0 |   g = 343331343
0   0   0   0   0   0   0   0 |   h = 3862712086
```

Figure 6.5: Confusion matrix of the tested data produced by Weka.

Used metrics and algorithms

For the evaluation of Context Player prediction model were as the metrics chosen metrics mentioned in 6.3, namely according to the assumption for the use of individual metrics were chosen Precision, Recall and overall F1 score metric.

Algorithms from section 6.4 were used in the application code. Both the train-test split and cross-validation were implemented using Weka library. The results of both variants are compared in this section.

To set up the specification of the random forest model parameters, each random forest had 100 trees and each tree was constructed considering 5 randomly picked features.

The first metric was accuracy. The accuracy of a model was acquired from printed algorithm's results. The value was displayed in the following form together with the number of tested instances:

```
Correctly Classified Instances         609                 98.0676 %
```

The precision and recall were acquired the similar way as an accuracy from the algorithm statistics. As the multiclass version was used the Weighted F1 metric, which takes into account the number of samples in each class presented. The data were represented in the following form:

```
        Precision   Recall  F-Measure   ROC Area  Class
        0.923       0.98    0.95        0.904     0.994  0.993  2046003820
        0.98        0.926   0.952       0.904     0.994  0.995  3862712086
 Avg.   0.953       0.951   0.951       0.904     0.994  0.994
```

6.6 Evaluation results

In this section are summarized the acquired results of the model evaluation. On the model with the highest number The evaluation was run the Cross-validation with 10 folds. The evaluation process was conducted on the model trained with 1436 individual input context data samples. The first part of evaluation was conducted with changing values of number of trees and the number of randomly selected features. All the measurement results are displayed in the table in appendix A. The table 6.1 displays the final most significant measurements from the table in the appendix A. In each row are marked bold the crucial values that decided that the parameters are on the best setting. The first row displays the model metrics with the lowest number of incorrectly classified instances. The second and third row both shows the model settings with the lowest out-of-the-bag error. The only difference between the second and the third row is that the third row takes much longer time to build the model (1.3 s) which makes the second row the better result for setting the model parameters.

Trees [count]	**Features** [count]	**OOB error**	**RMSE**	**True Class** [%]	**Incorrect** [count]	**Model build time**[s]
40	16	0.0014	0,0156	99.72	**4**	0.44
80	16	0.0007	**0,0149**	99.58	6	0.83
160	8	0.0007	**0,0149**	99.58	6	1.30

Table 6.1: The section of Table in appendix A. It includes the best evaluated results based on the lowest RMSE and the number of incorrectly classified instances.

The second part of evaluation included comparing changing tree depth with the number of incorrectly classified instances. The number of trees was in this case constant and it was 40 trees. The comparison was made for all previously tested numbers of features. From Figure 6.6, it is possible to see that the low error rates are achieved most quickly with the 16 and 20 features with the tree depth of 4. For a lower number of randomly selected features, the higher depth is required and the number of incorrectly classified instances is stabilized after the tree depth of 6.

The third part included comparison of the train-test split and OOB error to detect model overfitting. The overfitting detection is based on the comparison of error calculated on the training data only (OOB) and the error computed from the testing dataset evaluation (train-test split) error. If the value of OOB error is much smaller then the train-test split error that the probability of the model overfitting is high. The value of RMSE of the train-test split evaluation was 0.0072 tested on the random forest with 40 trees, 16 random features and maximum depth set to 6. The OOB result was with the same parameters 0.0014. From this value it is possible to say that the model is on the edge of a possible overfitting.

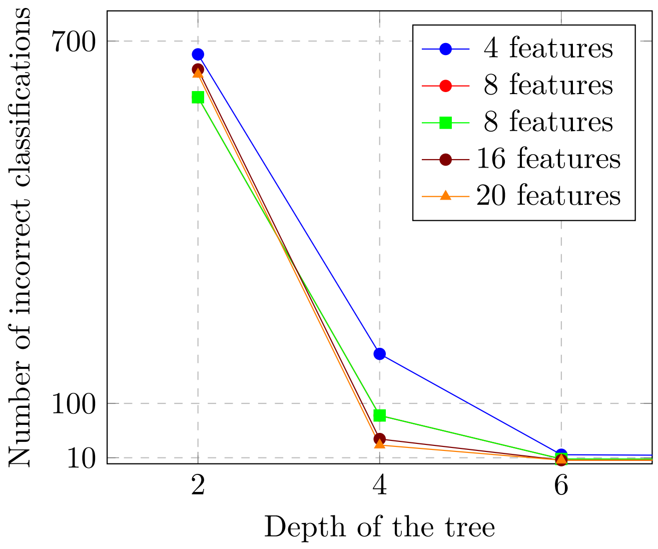


Figure 6.6: The graph displaying the dependency between the depth of the individual tree from random forest and the number of incorrectly predicted samples.

6.7 Evaluation of user models

In this section are discussed the test results. After collecting the data and performing the evaluation on the user models. The following results were acquired. The evaluation results are summarized in the following table:

Model	**OOB**	**Accuracy**	**Precision**	**Recall**	**F1**
Model 1	0.0102	99,15	0,99	0,99	0,99
Model 2	0.0135	99,03	0,99	0,99	0,99
Model 3	0.1168	88,62	0,89	0,89	0,89
Model 4	0.0007	99,58	1,00	1,00	1,00

Table 6.2: Table of calculated evaluation metrics on the tested user models.

From Table 6.2 is possible to observe that all tested models reported good results. In case of accuracy, it is not telling metric if the model is good or not if the data structure is not know (the number of samples of each class in the dataset). If this information is known and the data are balanced, and thus the accuracy metric is good choice. In this case the datasets were balanced thus presented accuracy can be used as the evaluation measure. The out-of-the bag error rates were in acceptable ranges too. All other metrics, such as F1-score, Precision and Recall, showed high values, around 0.99, where 1 is the maximum, which is common in Random Forest results.

6.8 Results discussion

The result of the evaluation are discussed in this section. It includes the interpretation of measured values and further recommendation for model parameters setting.

The random forest provide extremely high accuracy with about 99.9 % of correctly classified samples. This was in the case that the trees are allowed to grow to their maximum depth. The square-root of the mean-squared-error is 0.0187 in the worst case and 0.0149 in the best case. The estimation of out-of-bag error is in best case 0.0007, which is satisfactory value which stabilizes with the 80 or more trees used in model.

The best setting of the random forest parameters depending on the minimal RMSE are use of the 80 trees with selection of 16 random features in each tree which results in error value of 0.0007. The model will be built in this case for about 0.83 s. If targeting the lowest number of incorrectly classified instances, the 40 trees with 16 random features selected is the best setting resulting with 4 incorrectly classified instances from the total of 1436. The model will be in this case trained for 0.44 s.

The overall observation implies that the increasing number of trees results in reducing the OOB error. In the model evaluation the significantly better results of OOB error were obtained with 40 or more decision trees in the model.

In case of the tree depth, it was observed the the setting of the depth of the individual trees is important. The provided results show that the lower depth (around the 2 to 4) results in high number of inaccurately classified instances. On the other hand the higher depth then 6 is not necessary because the number of inaccuracies stabilizes after this threshold and it is not improving with the higher depth of the trees. With the tested setting of 40 trees, the accuracy in case of 4 random features selected was 52.79 %. The similar result is for other number of tested features. In case tree depth of 6 or more, the accuracy is about 98.96 % for 4 selected features.

The number of selected features is also important to consider. From Table in appendix A is possible to see that after reaching the minimal value, with increasing of the feature number the number of incorrectly classified instances rises. In the evaluated model the ideal number of selected are 8 or 16, which produces the smallest number of incorrect classifications.

The last test was testing the possible model overfitting. The testing of the OOB and train-test split error comparison concluded that there is a possibility of model overfitting.

The testing of the prediction model on real devices proved to work reliably accurate. The evaluation of the tested models also proved the correctness of the models. The high values of metrics such as accuracy and F1-score shows that the model should be able to find high number of correct classes and low of the wrong ones.

6.9 Publishing on Google Play

The application has been published on the Google Play store as a free application. Publishing of application on Google Play requires number of steps. At first it is required to specify the information about the application such as the type, target age group, verify compliance with the GDPR rules and fill in the questionnaire about the application content. In case of target group, the Context Player application could be targeted only from age of 13 because to support lower age the application has to include the privacy policy statement which should consult the location and other data collection in the application. GDPR also applies on location data in case that the person is identifiable by the location data, which is not the case of the Context Player application.

Afterward, the application presentation material needs to be crated. This includes creating the icon, preparing a set of screenshots and pictures that will be presented on the application store page or in an advertisement. Also a description text needs to be prepared for including in the store page.

After preparing of all the materials the application itself needs to be prepared for release. There are many guidelines which say how to modify the application itself to be suitable for release. The general rules are following. The first rule is to delete all files that are not actively used by the application, such as resources or classes that are no longer used. The second rule is to use *release* build variant of the application instead of the *debug*. This has to be done from two reasons, the first is to disable all the logging done by the application and the second is that the Google Play does not support uploading debuggable applications. The application then needs to be signed with the generated key.

After setting the release version of the application, the Android Studio offers to generate the Android Package Kit (APK) file, which is file used for application installation. This signed APK file is then loaded to the Google Play console and the application is ready to be published. The published Context Player application is available on the Google Play store[1].

In case of Context Player there were created a second version used for publishing in Google Play store. This version have longer intervals between the context data measuring, more specifically it measures the values after 10 seconds from the beginning of the song playback and then after each 40 seconds that the song is playing. The second thing that changed in comparison to testing version is use of the prediction trigger constraints which checks if the current context changed. The current context data are always on saving the data to file written to file accessed by *SharedPreferences* object, which preserves the current values even if the application was shut down. The data are retrieved on the check request and compared to the current context values. The detected variables which change triggers the prediction are:

- activity
- detection if the device is lying
- detection of the connected headphones
- change in connected WiFi name
- change in internet connection type
- device connected to charger
- type of charger changed

When one of those events occurs between the predictions, then it is detected on the next prediction job run. If at least one the variables changed, then the prediction will be triggered. The next value that changed in comparison to test version of the app is interval between the individual prediction jobs. In the test version they are triggered every minute but in the release are set to trigger every 10 minutes. The last aspect that is different from testing version is that was removed the button to send the collected data. This is because the sending is done via third party email application, thus the recipient can be changed and data can be easily downloaded by anybody. Data should not be possible to read by everybody or send to anybody, which could happen also by an accident. The second reason is to prevent spamming of the emails by a potential users.

[1]`https://play.google.com/store/apps/details?id=com.gabchmel.contextmusicplayer.release`

6.10 Publishing on GitHub and possible extensions

The application was published on as an open source to the GitHub platform. This platform was also used to version the code. In this section are also discussed the possible extensions of the application functionality and the prediction model.

Publishing as open source

The GitHub was chosen as a platform for sharing the solution as open source. The main reason to choosing this platform was that the application code was already published on such platform for the need of version the code. This platform was also used to keep track with issues and different versions of the code. There are two branches with two main versions — *dev* branch with the *debug* version and *master* branch with the *release* version. One of them was used for testing and the other one for publishing to Google Play store.

The published code also includes screenshots of the application. In this form can application reach more potential users and be more useful tool for music play automation. The link to the GitHub repository is available here [2].

Publishing the model conversion files

As was mentioned in Section 5.6, there were many techniques tested on how to deploy Random Forest machine learning model on Android device. The working example was created for the conversion from the Random Forest model written in Python with the scikit library to the pmml format. The code to make predictions with the converted pmml model on Android was published on GitHub[3].

6.11 Possible application extensions

There are many aspects in which can be the application extended. Some of the ideas that were not finished but would be good to implement in further versions are mentioned in this section.

- *Application UI* — In case of the application UI there is always space for improvement. There can be animations added, personalized view of the list, the possibility of language change and so on.
- *Location clustering* — In model prediction part there were proposed many improvements that could be implemented in the later versions. First proposed improvement would be to cluster the locations into the general groups, such as home (group num. 1), work (group num. 2), etc. This would be achieved and is already implemented with the DBSCAN clustering algorithm. The reason this solution is not used is that in the algorithm is allocated matrix with all possible distances, which cannot be allocated with bigger dataset (hundreds of values) at once.
- *All data anonymization* — The second proposed optimization would be to hash all the data (currently on WiFi name is) to provide more security. This is not done now because currently the application does not need internet connection to run thus it is not necessary.

[2] `https://github.com/4Gabby4/context-player`
[3] `https://github.com/4Gabby4/pmml-android`

Chapter 7

Conclusion

In the first part of the thesis, the goal was to present the current state of context-aware applications and explore possible ways of detecting context of a mobile device. The second part dealt with a discovery of possible ways of distinguishing different contexts of the device using machine learning. The third part proposed a design of the music player application that used machine learning for the context classification.

An overview of developing a multimedia mobile application for Android operating system was proposed. Then the context of the device was introduced, which can be derived from the physical and virtual data, and ways of context detection on Android OS. Frameworks were introduced that simplify access to sensors and device's virtual background data. Lastly, a few applications that use context of the device were showcased.

The basic machine learning approaches were introduced and methods of classification were described. Then, the frameworks that can help to implement machine learning model on a mobile device were named.

Then in the design chapter the application user interface was described together with the feature specification. Moreover, a machine learning algorithm was chosen to later classify the different context situations of the device.

In the chapters dealing with the implementation of the Context Player was described the implemented solution, together with description of the machine learning prediction process. The last chapter described the testing of the implemented application and evaluation of it's machine learning model. This chapter also includes the discussion of the evaluation results. The application was successfully implemented and its model was proved to be working. The resulting solution was published as an open source to the GitHub platform and on the Google Play store.

Bibliography

[1] ABOUT O'REILLY. *Mean Squared Error and Root Mean Squared Error* [online]. Jul 02, 2021. [retrieved 2021-07-02]. Available at: `https://www.oreilly.com/library/view/machine-learning-with/9781785889936/669125cc-ce5c-4507-a28e-065ebfda8f86.xhtml`.

[2] AGARWAL, R. *The 5 Classification Evaluation metrics every Data Scientist must know* [online]. Sep 17, 2019. [retrieved 2021-07-04]. Available at: `https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226`.

[3] ALAM, M. S. and VUONG, S. T. Random Forest Classification for Detecting Android Malware. In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 2013, p. 663–669. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.122.

[4] ARUNKUMAR. *Conscient - Context Aware app* [online]. Aug 23, 2016. [retrieved 2021-07-03]. Available at: `https://play.google.com/store/apps/details?id=arun.com.jarvis`.

[5] AVRAM, A. *Funf Is a Sensing and Data Processing Mobile Framework* [online]. Jan 14, 2012. [retrieved 2021-01-07]. Available at: `https://www.infoq.com/news/2012/01/Funf/`.

[6] AWARE. *Google's Play Store* [online]. Feb 27, 2020. [retrieved 2021-01-05]. Available at: `https://awareframework.com/`.

[7] BALTRUNAS, L., KAMINSKAS, M., LUDWIG, B., MOLING, O., RICCI, F. et al. InCarMusic: Context-Aware Music Recommendations in a Car. In: HUEMER, C. and SETZER, T., ed. *E-Commerce and Web Technologies*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, p. 89–100. ISBN 978-3-642-23014-1.

[8] BROWNLEE, J. *A Gentle Introduction to k-fold Cross-Validation* [online]. May 23, 2018. [retrieved 2021-07-15]. Available at: `https://machinelearningmastery.com/k-fold-cross-validation/`.

[9] CHARFAOUI, Y. *CWorking with Geospatial Data in Machine Learning* [online]. Nov 13, 2019. [retrieved 2021-06-12]. Available at: `https://heartbeat.fritz.ai/working-with-geospatial-data-in-machine-learning-ad4097c7228d`.

[10] CHAUHAN, N. S. *Random Forest — A Powerful Ensemble Learning Algorithm* [online]. Jan 20, 2020. [retrieved 2021-07-15]. Available at: `https://www.kdnuggets.com/2020/01/random-forest-powerful-ensemble-learning-algorithm.html`.

[11] d'wise one. *What's A Gyroscope And Accelerometer Doing In My Mobile Device?* [online]. Jun 30, 2015. [retrieved 2020-11-26]. Available at: https://medium.com/chip-monks/whats-a-gyroscope-and-accelerometer-doing-in-my-mobile-device-eb7acbdfa4e0#:~:text=A%20gyroscope%20in%20your%20phone,of%20reference%20of%20the%20device.

[12] Elmalaki, S., Wanner, L. and Srivastava, M. CAreDroid: Adaptation Framework for Android Context-Aware Applications. *MobiCom '15*. 1st ed. september 2015, no. 1, p. 386–399. DOI: 10.1145/2789168.2790108.

[13] Ferreira, D. *Context* [online]. Feb 27, 2020. [retrieved 2021-01-05]. Available at: https://awareframework.com/context/.

[14] Google. *Sensors Overview* [online]. July 1993. [retrieved 2020-11-23]. Available at: https://developer.android.com/guide/topics/sensors/sensors_overview.

[15] Google. *Media app architecture overview* [online]. Feb 3, 2021. [retrieved 2020-04-15]. Available at: https://developer.android.com/guide/topics/media-apps/media-apps-overview.

[16] Google. *WorkManager basics* [online]. Jan 22, 2019. [retrieved 2021-05-10]. Available at: https://medium.com/androiddevelopers/workmanager-basics-beba51e94048.

[17] Google. *ConnectivityManager* [online]. Jun 9, 2020. [retrieved 2020-12-26]. Available at: https://developer.android.com/reference/android/net/ConnectivityManager#nested-classes.

[18] Google. *MediaPlayer overview* [online]. Mar 01, 2020. [retrieved 2020-11-25]. Available at: https://developer.android.com/guide/topics/media/mediaplayer.

[19] Google. *Meet Android Studio* [online]. May 18, 2020. [retrieved 2020-11-23]. Available at: https://developer.android.com/studio/intro.

[20] Google. *Bluetooth overview* [online]. May 19, 2020. [retrieved 2020-12-27]. Available at: https://developer.android.com/guide/topics/connectivity/bluetooth#HDP.

[21] Google Developers. *Machine learning for mobile developers* [online]. Jan 07, 2021. [retrieved 2021-01-07]. Available at: https://developers.google.com/ml-kit.

[22] Google Play. *Automate* [online]. Jan 6, 2021. [retrieved 2021-01-06]. Available at: https://play.google.com/store/apps/details?id=com.llamalab.automate&hl=en&gl=US.

[23] Goyal, K. *Machine Learning vs Neural Networks: What is the Difference?* [online]. Feb 13, 2020. [retrieved 2021-01-07]. Available at: https://www.upgrad.com/blog/machine-learning-vs-neural-networks/#:~:text=Machine%20Learning%20uses%20advanced%20algorithms,modelling%20using%20graphs%20of%20neurons.

[24] Great Learning Team. *Random Forest Algorithm - An Overview* [online]. Feb 19, 2020. [retrieved 2021-07-15]. Available at: https://www.mygreatlearning.com/blog/random-forest-algorithm/.

[25] GSMArena. *Sensors - definition* [online]. Nov 29, 2020. [retrieved 2020-11-29]. Available at:
https://www.gsmarena.com/glossary.php3?term=sensors#:~:text=The%20digital%20compass%20that's%20usually,depending%20on%20your%20physical%20orientation.

[26] h4h13. *Retro Music Player* [online]. Oct 12, 2020. [retrieved 2020-11-20]. Available at: https://github.com/h4h13/RetroMusicPlayer.

[27] Ham, N., Dirin, A. and Laine, T. H. Machine learning and dynamic user interfaces in a context aware nurse application environment. *Journal of Ambient Intelligence and Humanized Computing*. 1st ed. april 2017, vol. 8, no. 2, p. 259–271. DOI: 10.1007/s12652-016-0384-1.

[28] Hao, K. *What is machine learning?* [online]. Nov 17, 2018. [retrieved 2021-01-07]. Available at: https://www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart/.

[29] HUAWEI Technologies Co. *Sensors on a phone/tablet and their functions* [online]. Nov 25, 2020. [retrieved 2020-11-25]. Available at:
https://consumer.huawei.com/eg-en/support/content/en-us00685236/.

[30] Kacz, K. *Context Aware Android Application Trace Analysis*. 2013. Master's thesis. Charles University in Prague.

[31] Lacko Ľuboslav. *Vývoj aplikací pro Android*. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

[32] LlamaLab. *Automate* [online]. Jun 22, 2021. [retrieved 2021-07-05]. Available at:
https://play.google.com/store/apps/details?id=com.llamalab.automate&hl=en&gl=US.

[33] Lyashenko, A. V. *Cross-Validation in Machine Learning: How to Do It Right* [online]. Jun 15, 2021. [retrieved 2021-06-20]. Available at: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right.

[34] Lyashenko, V. *Cross-Validation in Machine Learning: How to Do It Right* [online]. Jun 19, 2021. [retrieved 2021-07-10]. Available at: https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right.

[35] Mikulski, B. *How to deal with days of the week in machine learning* [online]. Mar 26, 2021. [retrieved 2021-03-30]. Available at:
https://www.mikulskibartosz.name/time-in-machine-learning/.

[36] Milić, E. and Stojanović, D. EgoSENSE: A Framework for Context-Aware Mobile Applications Development. *Engineering, Technology & Applied Science Research*. 1st ed. august 2017, vol. 4, no. 4, p. 1791–1796. DOI: https://doi.org/10.48084/etasr.1203.

[37] Mohajon, J. *Confusion Matrix for Your Multi-Class Machine Learning Model* [online]. May 29, 2020. [retrieved 2021-07-18]. Available at:
https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826.

[38] Ogbo, O. *Using the Awareness API in your Android app* [online]. Jul 25, 2016. [retrieved 2021-01-04]. Available at: `https://www.androidauthority.com/using-awareness-api-705176/`.

[39] Ogbo, O. *Using the Awareness API in your Android app* [online]. Jul 25, 2016. [retrieved 2020-11-30]. Available at: `https://www.androidauthority.com/using-awareness-api-705176/`.

[40] Osiński, B. and Budek, K. *What is reinforcement learning? The complete guide* [online]. Jul 5, 2018. [retrieved 2021-01-07]. Available at: `https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/`.

[41] Ravindran, R., Suchdev, R., Tanna, Y. and Swamy, S. Context aware and pattern oriented machine learning framework(CAPOMF) for Android. In: *2014 International Conference on Advances in Engineering Technology Research (ICAETR - 2014)*. 2014, p. 1–7. DOI: 10.1109/ICAETR.2014.7012912.

[42] Salian, I. *SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning?* [online]. Aug 2, 2018. [retrieved 2021-01-07]. Available at: `https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/#:~:text=In%20a%20supervised%20learning%20model,and%20patterns%20on%20its%20own`.

[43] Schilit, B., Adams, N. and Want, R. Context-Aware Computing Applications. In: IEEE, ed. *1994 First Workshop on Mobile Computing Systems and Applications*. 1994, p. 85–90. DOI: 10.1109/WMCSA.1994.16. ISBN 978-0-7695-3451-0.

[44] Scikit learn. *Support Vector Machines* [online]. Jan 01, 2021. [retrieved 2021-01-07]. Available at: `https://scikit-learn.org/stable/modules/svm.html`.

[45] Yiu, T. *Understanding Random Forest* [online]. Jun 12, 2019. [retrieved 2021-07-18]. Available at: `https://towardsdatascience.com/understanding-random-forest-58381e0602d2`.

[46] Zhang, D., Adipat, B. and Mowafi, Y. User-Centered Context-Aware Mobile Applications—The Next Generation of Personal Mobile Computing. *Communications of the Association for Information Systems*. 1st ed. december 2009, vol. 24, no. 2. DOI: 10.17705/1CAIS.02403.

Appendix A

Random forest model evaluation results

Trees [count]	**Features** [count]	**OOB error**	**RMSE**	**True Class** [%]	**Incorrect** [count]	**Model build time**[s]
10	4	0.0669	0,0187	994.429	8	0.05
10	8	0.0682	0,0172	995.822	6	0.08
10	16	0.0662	0,0172	995.125	7	0.1
10	20	0.0675	0,0177	993.733	9	0.12
20	4	0.0077	0,0175	994.429	8	0.11
20	8	0.0084	0,0158	995.822	6	0.14
20	16	0.0084	0,016	995.822	6	0.22
20	20	0.0077	0,0165	995.822	6	0.24
40	4	0.0007	0,0167	994.429	8	0.21
40	8	0.0014	0,0153	995.822	6	0.32
40	16	0.0014	0,0156	997.214	**4**	0.44
40	20	0.0014	0,0167	995.822	6	0.46
80	4	0.0007	0,0156	995.822	6	0.4
80	8	0.0007	0,015	995.822	6	0.56
80	16	0.0007	**0,0149**	995.822	6	0.83
80	20	0.0007	0,0161	995.822	6	1.02
100	4	0.0007	0,0157	995.822	6	0.63
100	8	0.0007	0,0151	995.822	6	0.72
100	16	0.0007	0,015	995.822	6	1.25
100	20	0.0007	0,0163	995.822	6	1.31
160	4	0.0007	0,0156	995.822	6	0.8
160	8	0.0007	**0,0149**	995.822	6	1.3
160	16	0.0007	0,0153	995.822	6	1.85
160	20	0.0007	0,0164	995.822	6	2.06

Table A.1: The table displays the measured evaluation metrics. The bold values in the *RMSE* and *Model build time* columns represent the best measured values of the evaluation.

Appendix B

Contents of the included storage media

```
/ ................................................. root directory of the included CD
├── src. .......................................... source code of the application
├── text
│   ├── xchmel27-Android_context_player.zip ......... source files of the thesis text
│   └── xchmel27-Android_context_player.pdf ............. thesis text in pdf format
├── datasets. ................................ folder with a table of collected data
├── executables. ................................ folder with executable APK files
│   ├── 01. ...................................... folder with the data from tester 1
│   ├── 02. ...................................... folder with the data from tester 2
│   ├── 03. ...................................... folder with the data from tester 3
│   └── 04. ...................................... folder with the data from tester 4
├── PredictionModelpmml.kt. ..................... implemented pmml conversion file
└── screenshots. .......................................... application screenshots
```