Gaon – Movement Indication Module

Graduation Report

Tiemen Wierda

June 2023

Version History

Version	Date	Description of changes
0.1	17-Feb-2023	First version
0.2	18-Apr-2023	Changed document layout to be in line with other project documentation.
0.3	19-Apr-2023	Changed all chapters to be more detailed and in line with the style of this document, added phase 3
0.4	15-May-2023	Processed feedback, some rewrites, continued reporting on latest developments, including but not limited to machine learning, research, and app development
1.0	05-Jun-2023	Concept version of the Final Delivery. Processed feedback, added result, competences, and several realization chapters.
1.1	17-Jun-2023	Final delivery version with processed feedback: • Added page on stakeholders • Added discussion page • Moved Professional Competences to Personal Reflection document • Applied more consistent terminology • Several rewrites, additions, and readability changes

Table of Contents

Introduction

Disclaimer

This report serves as documentation for the entire graduation process. I will detail every step chronologically, discuss results and give feedback. This means that the documentation of earlier parts may make assumptions that are not necessarily executed in that manner in later parts. This document is written as the project went on, and the contents will reflect that.

The goal of this document is to give a full report of everything I have done. There will be other documentation, in which some details are handled, but none should be necessary reading material in order to follow along with this document.

Brief introduction

User data is valuable. This is the main theme that led Baseflow to the creation of this project. Baseflow, a software-development company in Enschede wishes to add a method of collecting data to the 'Gaon'-app: a traveling app currently in development. The goal of this project, then, is to create a method to classify the mode of transportation of the users of Gaon, and make this data available in several ways. From raw data, to visual interpretations of this data, such as an algorithm that presents users with their route history, or a heatmap detailing where exactly people frequently travel with certain types of transportation. Baseflow has set very few requirements, and left almost all of the details of the project to me.

Reading Guide

This project is split into several phases, or overarching chapters. Phase 1 gives an overview of the project details. It discusses what the project is about, how to solve the problems as described in the project description, what kind of software implementation will be built, and the requirements of this system.
In Phase 2, the initial research steps are detailed. It first describes the research questions and the research approach, and then answers the research questions necessary to build the first prototype. A detailed report of the development of this app prototype is given in Phase 3. This app prototype Is responsible for collecting a dataset, which is then used to answer the research questions that rely on it in Phase 4.
The development of the final products, excluding the data collection app, is detailed in Phase 5. It handles the functionality and steps taken to build these products. Lastly, the results of the development phase are discussed; which parts were built, and to which degree they satisfy the requirements.

Phase 1: Orientation & Background

Orientation is about finding out where you stand. This is also what the contents of this phase are about. This phase began before the project officially started, by discussing details with Baseflow and laying out the project description. From there, the first couple weeks of the project were spent getting used to Baseflow and getting the plan of approach ready.

In this graduation report, the first phase details many points in the organization of this project. It includes a detailed background, methodology, requirements, stakeholders, and more. Most important background information relating to the product itself are described in this chapter. For further information on risk assessment, project management, and scheduling, the Plan of Approach itself should be consulted.

Project Description

This project has been proposed by Baseflow. Baseflow is a software development company in Enschede. It is a fairly new company, at 5 years old. They develop custom software for clients, and some products to market themselves.

Baseflow is currently developing a mobile app named Gaon. The main goal of this app is to provide a travel route from a to b, using a broad scale of different regional and local travel options, such as walking, loaner bikes, shuttle busses, rental cars, but also riding along with individuals, as well as the more traditional busses and trains. Alpha versions of this app are being tested by a small group of users. For the scope of this project, however, the status of the Gaon app's development is irrelevant.

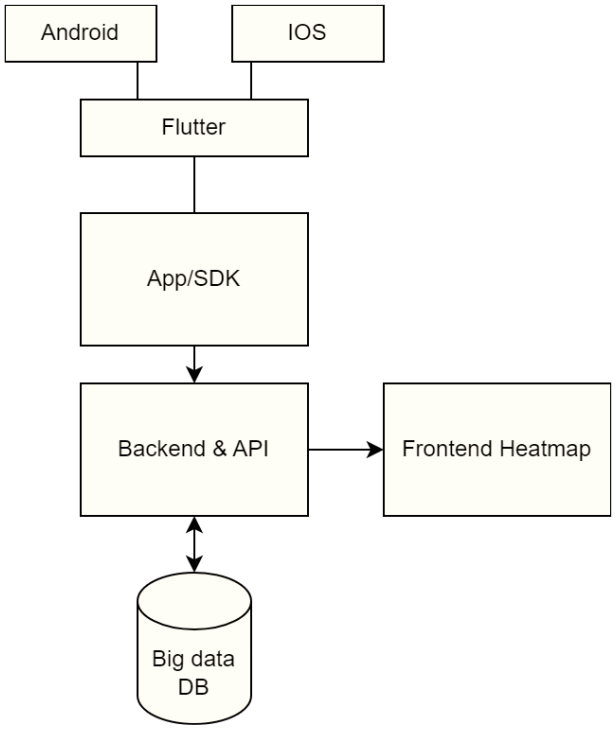

Figure 1: Diagram of a hypothetical solution brainstormed during an early meeting

This project originated from a desire by municipalities to have insight in user travel statistics, as well as a desire from Baseflow to collect some meaningful data from users. Data concerning more niche forms of travel is a valuable source for statistical analysis. Some competitor apps already provide a version of this data, but Baseflow wants to supply this data as part of their own application.

The Gaon app, which is maintained by a separate team at Baseflow, is not processing any user data, while there is a lot of potentially useful data able to be collected. This meant that a new software solution needed to be built, with the express purpose of collecting data on modes of transportation and make them available.

The primary goal of this project is to write software that collect and classifies data about a user's trip, to later be used for statistical analysis. This software module is intended to run in the background of the

Gaon app. There are no existing parts already available, which means that I will have to develop everything myself. The software that is to be built needs to fulfil these two primary requirements:

- To predict, for every point in a user's trip, what kind of transit option the user is taking, be it a bike, bus, car or train.
- To keep track the exact route a user has taken. This does not refer to the route a user has planned. That specific data is not available for this project. Instead, the to be built software solutions need to collect data, and reconstruct the taken route.

The main goal of this project, then, is to provide this data. The data will be used in future projects, and potentially during this project in the form of a heatmap. The target group of the to be provided data are municipalities, but potentially also transport providers and end users.

Challenges

Part of the project are several expected challenges. These challenges are potentially complex parts of the project, based on requirements by the product owner.

- The database needs to be able to process and store large amounts of data. Each user produces many datapoints per trip, and the backend needs to be able to accommodate hundreds of concurrent users.
    - Given the large data flow, the per-point prediction needs to be cheap in terms of processing power required.
- The classification algorithm needs to predict exactly what type of travel option a user is using at every datapoint, as accurately as possible. This is particularly challenging since there are a large amount of real-world edge cases to consider. A couple examples of these edge cases are:
    - Speed is a metric that can serve as a breakpoint for certain modes of transportation. A cyclist will never reach the speed of a moving train, for example. It is not a catch-all solution, however. Speed can vary greatly at different points in someone's trip, so how can the algorithm avoid identifying fast-moving slow vehicles as slow-moving fast vehicles, and vice-versa?
    - When standing still at a multitype traffic light junction, since there cannot be a prediction based on speed or location, how can we accurately predict the type of transport?
    - If a train track is next to a highway, and the GPS is within margin of error, how can we safely differentiate a fast-moving car from a slow-moving train?
- The data collection needs to be minimally invasive for users. This means that beyond consenting to the data collection, the collected data needs to be small enough as to not incur a major cost to the user's data budget. Additionally, the on-device module should not pose a significant drain on the user's battery.

Project Details

The project consists of several parts that were designed and built. These are:

- An on-device module. This is a Dart library that can be implemented in any app. The library provides API methods to allow developers to collect labeled user data.
- A data collection app. In order to start training a machine-learning algorithm, data is needed. This app will be built to fulfill that need. It will have much of the functionality of the on-device module.
- A classification algorithm. This is the algorithm responsible for classifying the user's mode of transportation. It will work by training a machine-learning model on a test set of data, which aims to result in an as high as possible confidence rating. This algorithm will run on the user's device if possible. Otherwise, a backend server or cloud implementation will be built to host this algorithm.
- A database. This database serves to collect all datapoints collected by users.

The following parts are of a lower priority. They were potential parts of the project, meant to serve as possible extensions for if the above parts are completed. In the end, the prototype described at first bullet point below has been completed.

- Another algorithm needs to be written that collects a specific dataset from the database, and reconstructs a users' taken route.
- A heatmap. This is another potential expansion of the scope. This heatmap will give insight into the collected data. Users will be able to see where certain modes of transportation are in high usage, and compare this data against other modes of transportation.

Preconditions

Beyond the requirements I gathered from my requirement-analysis, a small number of preconditions have been set by the project supervisor. Some of these have already been touched on.

- The final implementation has to be usable in both the Gaon app, as well as other apps. Gaon is made in the Futter framework, so the on-device module needs to be built to be implementable in any Flutter project.
- Privacy of the user needs to be taken into account.
- The battery usage of user's phones should be minimally impacted.

What this means in practice is that the on-device module will be built as a library, so that it can be shared and implemented in flutter apps such as Gaon. The privacy concern will be thoroughly researched, and code efficiency will be kept in mind to minimize battery usage.

Project Stakeholders

Student

I am the main executor of the project. Outside of the project's definition and preconditions, all aspects, from planning, designing, to implementation, are performed by me.

Saxion Supervisor

The supervisor from Saxion, Kevin Wilmink, is the main point of contact from Saxion. He keeps an eye on whether the graduation process is proceeding well, and is available for contact if I have questions regarding the graduation process. We plan a monthly meeting to discuss these items. He receives deliverables at set deadlines, and give feedback based on these deliverables. He is also responsible for giving 'go' or 'no-go' signals in the final phase of the project.

Baseflow Project Supervisor

The project supervisor, Jorrit Everts, is the main point of contact for me when I have questions. They are also part of weekly meetings to discuss progress, and give feedback on work done.

Baseflow

Baseflow Is the product owner of the Gaon app- of which this project's result will be part of. In addition, baseflow provides a work environment, the resources necessary to work on the project, as well as guidance in the form of a company supervisor and other colleagues.

Municipalities

Municipalities are the primary target for sharing the data generated by the on-device module. It is therefore important to understand the desires of these municipalities so that the on-device module can meet them.

Gaon app users

The app users make up the group of whose data is being collected. For this group, it is important to minimize the impact that the on-device module has on the user experience.

Architecture model

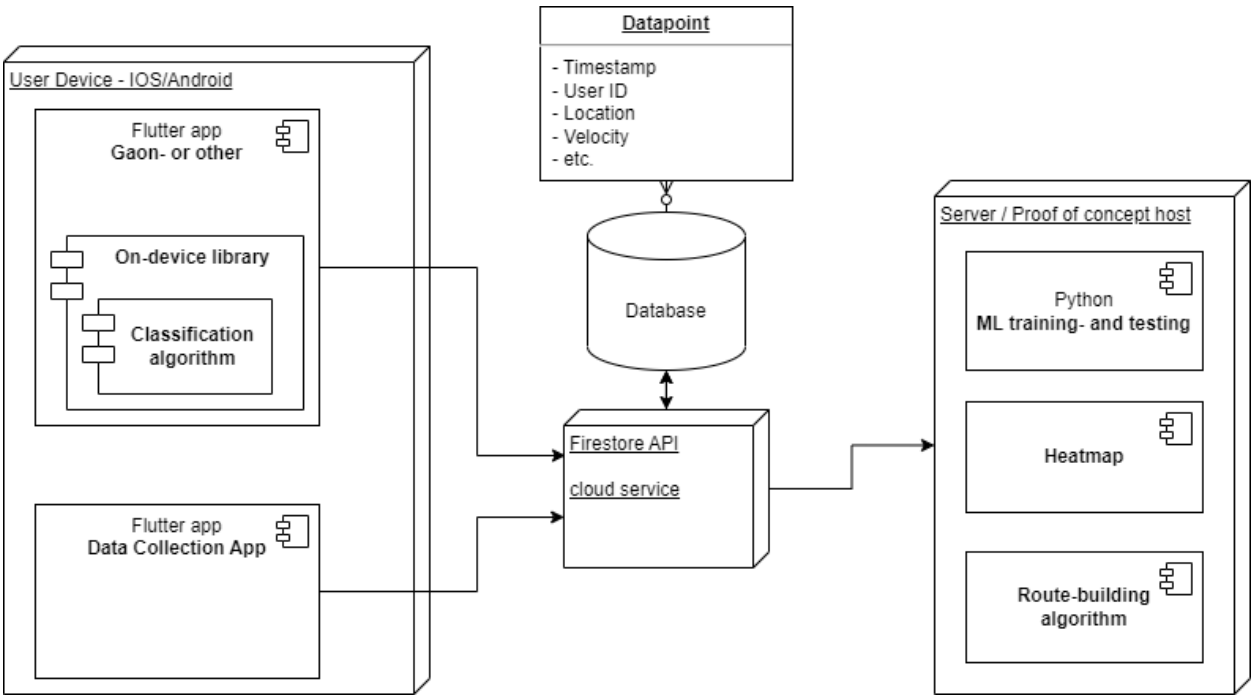

Figure 2: Software architecture model

This model shows all the different pieces of software, or prototypes, that I have been planned; where they will live, and how the data will flow. I do not need to deploy the code on a local- or cloud server; all algorithms on the right, in the Server category, will be manually run- locally hosted proof of concepts. The app and library on the phone do need to run, however.
The on-device module itself is not responsible for sending data to the database, it instead presents data to the app that implements the library, so that the app itself can chose how it wants to send it to any database it would like. In the above diagram, this is the same database as the Data Collection App. The data collection app directly connects to the database in order to send information.
The prototypes and PoC's (Proof of Concepts) on the server work based on a database snapshot system. Every so often, a snapshot of the most recent database entries is loaded via script onto the server. This process is not yet automated since it was unnecessary for the creation of the prototypes and PoC's. The route-building algorithm and heatmap are not supposed to be integrated in the on-device module, and are their separate PoC's. Their final implementation target has not yet been decided by the product owner.

Requirements

In collaboration with the company supervisor, the following requirements have been set up. These are split into several sub-projects, ordered by priority. The On-device module, Classification Algorithm, and the data collection app that that algorithm relies on, are necessary for the minimum viable product of this project and will therefore be prioritized. Once all of their requirements are fulfilled, the route building algorithm and heatmap sub-projects will be started. Lastly, if time remains, the frontend implementation will be considered.
Each list of requirements is accompanied by a short description of the to-be-built prototype. This list is in order of importance, mostly because the earlier parts are required for functionality in the later parts. As for priorities, the 'must' priorities need to be finished before moving on to developing the next prototype. Additionally, all 'should' priorities of the data collection app, classification algorithm and on-device module need to be finished before moving on to the heatmap or the frontend.

Data collection app

The data collection app is necessary for the classification algorithm.

Table 1: Data collection app requirements

Priority	Functionality	Description
Must	Functional	The data collection app needs to collect data in set intervals
Must	Functional	The data collection app needs to send all collected data to a remote database
Must	Non-functional	The data collection app needs to be written in dart and work in flutter
Should	Functional	The data collection app needs to collect all potentially relevant sensor data
Should	Functional	The data collection app needs to be able to collect data while running in the background
Should	Functional	The data collection should be toggleable by the user
Should	Functional	The user should be informed via notification while the data collection is happening in the background
Should	Functional	The user needs to be able to request deletion of data
Should	Non-functional	The data collection app should be made available for internal sharing via the Play store/Appstore
Should	Functional	The database should be query-able by date and device-id

Classification algorithm

The classification prototype is divided into two parts. The training, building and testing happens on the server- or proof of concept machine. The final classifier, then, is exported to the on-device module and used on-edge to classify datapoints.

Table 2: Classification algorithm requirements

Priority	Functionality	Description
Must	Functional	The algorithm needs to predict a user's mode of transportation based on collected datapoints
Must	Functional	The algorithm should account for potentially inaccurate datapoints
Must	Functional	The classification score should be as high as possible
Should	Functional	The algorithm needs to periodically retrain itself with new data, either manually or automatically

On-device module

The on-device module is what runs in the background of the app that implements it. It is responsible for the collection of data. These are largely similar to the data collection app, but since there is no implicit frontend, those requirements are omitted.

Table 3: On-device module requirements

Priority	Functionality	Description
Must	Non-functional	The on-device module needs to be as isolated as possible, as opposed to integrated
Must	Functional	The on-device module needs to collect data in set intervals
Must	Functional	The on-device module needs to send all collected data to a remote database
Must	Non-functional	The on-device module needs to be written in dart and work in flutter
Must	Functional	The data collection needs to comply with GDPR privacy laws
Should	Functional	The on-device module needs to be configurable by the implementer
Should	Non-functional	The on-device module needs to be as efficient in battery and data usage as possible
Should	Functional	The on-device module needs to be able to collect data while running in the background
Could	Non-functional	The on-device module needs to be made available as a flutter package

Route-building algorithm

The route-building algorithm will take datapoints, and construct one or several routes taken by specific users.

Table 4: Route-building algorithm requirements

Priority	Functionality	Description
Must	Functional	The backend needs to be able to construct a user's travel routes when given a set of datapoints.
Must	Functional	The backend needs to have functions to which datapoints can be entered and a user's travel history will be returned

Heatmap prototype

The heatmap prototype is another potential expansion of the project, on which collected data is displayed.

Table 5: Heatmap prototype requirements

Priority		Description
Must	Functional	The backend needs to host a heatmap of collected datapoints
Should	Functional	Users should be able to filter the heatmap by vehicle type
Should	Functional	Users should be able to select a timeframe to filter the heatmap by
Could	Functional	The heatmap should keep up to date with new live data

On-device module – Frontend

The frontend for the Gaon app, this is the least important, optional component. If it ends up being a part of this project, it needs to give users control of the on-device module's functionality while also making sure that GDPR is complied with.

Table 6: On-device module - Frontend requiremetns

Priority		Description
Must	Functional	The frontend needs inform users as to the nature of the data collection
Must	Functional	The frontend needs to confirm user consent
Must	Functional	Users need to be able to review collected data
Must	Functional	Users need to be able to enable and disable data collection
Must	Functional	Users need to be able to request deletion of collected data
Should	Functional	The frontend should notify the user while data collection is taking place

Phase 2: Research

In order to find the details of how certain parts of the on-device module need to be built, research has been done. The total research has been split into two phases. This is done to allow for data collection as early as possible. Several prototypes depend on data being available, so the first research phase will only answer the questions that are critical for the functioning of the data collection app.

Problem definition and primary research question

The Gaon app currently does not collect user data. Baseflow wishes to collect more than just basic data, however, and also wants to somehow determine the mode of transportation of users at any point in their journey, without users actively giving this information. Baseflow currently does not have the tools to either collect user data, or classify their mode of transportation.

From this problem definition, the primary research question can be formulated:

"How can I predict, with high accuracy, the mode of transportation of someone by using a phone's sensors, in order to present this in a readable way?"

Sub-questions:

- *What data can I collect from a user in order to be able to determine travel behaviour?*
- *What is Flutter and how can I use it to build a mobile application?*
- *How is GPS data processable?*
- *What are the user privacy concerns with collecting data and how can they be solved?*
- *Is it possible to employ Machine Learning in mode of transportation classification?*
- *How can an accurate prediction of vehicle type be made?*
- *What is the minimum acceptable quality of collected data, and how can phone data usage be optimized?*

These questions have been written to answer the first of the two main functional requirements: the classification of the mode of transportation of each data point. The second function; the reconstructing of travelled routes, is not the main functionality, and is significantly different to the point of it not fitting in the main research question. It is therefore separate, and described as follows:

"How can I correctly segment routes from a set of datapoints, and provide these routes on a visual map?"

Addendum: while this was a question at the start of the project, I overestimated its complexity, and the question has been answered during development. Each datapoint contains the necessary data to group connected datapoints together, and an existing library function can be used to chart these on a visual map. I will not dedicate a research chapter to explaining this in detail; this has been done in the realization phase of the route-building algorithm.

In order to research the primary research question and its sub questions, several methods of research have been utilized. Online literature has been the main source of information. For a general direction in which to do research, my own ideas and feedback from the project supervisor has been used. For practical questions, small prototypes might have been built.

Research Setup

Each research question will be executed in much the same way. The first step is to describe the problem in more detail. After this, research and ideation will be done for possible solutions, after which a solution is chosen and further researched. If needed, a simple, focussed, proof-of-concept is made.

1. What data can I collect from a user in order to be able to determine travel behaviour?

In order to determine travel behaviour, user data is needed. For location-based analysis, it is fairly obvious to use GPS data, but a phone has many usable sensors. What kind of data can be collected from a Flutter app?

Process and Solution

To understand what kind, and exactly what data will need to be collected, the end goal of the project needs to be considered. The main function of the on-device module is to simply collect data. This data needs to be labelled, ultimately without user input. In order to predict these labels, additional sensor data is likely useful. Additionally, the collected data needs to be usable for differentiating between users and chart all their taken routes.

To solve both of these requirements, a set of datapoints can be used, each representing a certain location at a certain point in time. These datapoints then contain a number of metrics, such as user information and sensor data. These datapoints can then be used to either process a predicted mode of transportation, or, by using a collection of datapoints from the same user, chart a route.

The following table details the most commonly available phone-based sensor types as described in official documentation by Google and Apple[1]. I then cross-referenced these sensors with available Flutter packages[2] to see which ones I can implement in that framework. Using Flutter- or at least Dart (Flutter's language), is a base requirement for the module, so the sensors need to be available in this framework. The table is then filled in to show their availability and projected usefulness to the needs of the on-device module.

[1] Sensors Overview. (n.d.). *Android Developers*. https://developer.android.com/guide/topics/sensors/sensors_overview

Sensor types. (n.d.). *Android Open Source Project*. https://source.android.com/docs/core/interaction/sensors/sensor-types

SensorKit | *Apple Developer Documentation*. (n.d.). Apple Developer Documentation. https://developer.apple.com/documentation/sensorkit

[2] *sensors_plus* | *Flutter Package*. (n.d.). Dart Packages. https://pub.dev/packages/sensors_plus

Prasad, A. (n.d.). *Top Flutter Motion Sensor*, *Light Sensor*, *Compass*, *Accelerometer*, *Gyroscope packages* | *Flutter Gems*. Flutter Gems - a Curated List of Dart & Flutter Packages. https://fluttergems.dev/sensors/

Table 7: Availability and suitability of most common phone sensors

Sensor	IOS	Android	Flutter	Provides possible metric for travel history	Provides possible metric for mode of transportation
GPS	X	X	X	X	X
Accelerometer	X	X	X	-	X
Gyroscope	X	X	X	-	?
Magnetometer	X	X	X		?
Ambient temperature	-	X	X	-	-
Heart Rate	?	?	-	-	-
Light	X	X	X	-	-
Proximity	X	X	X	-	-
Pressure	X	X	X	-	-
Humidity	X	X	X	-	-

One thing I noticed from researching this is the general availability of phone sensors. I am unsure what I expected, but there are fewer sensors overall than I initially thought. There also are only a few reliably available on Flutter, and I have to consider the compatibility between devices. It will be difficult to start collecting data from sensors that only have a 50% user base, as they will negatively impact the classification process.

The main potential issue with these sensors is that not all devices may include every sensor. Filtering out sensors that are unlikely to contribute relevant metrics, sensors that are unavailable on Flutter or one operating system, and sensors that are niche enough to potentially be unavailable on a majority of phones, a few metrics remain:

- GPS
    - The most obvious and important metric, positioning data, serves as the baseline for the functionality of nearly every prototype.
- Timestamp
    - In order to keep track of a user's route, timestamps are necessary. Additionally, they could be used to generate sensor data over time, or deltas, which can improve the predictive algorithm. Timestamps are also easy to add.
- Accelerometer
    - Less useful for travel history, knowing an approximate value for user velocity will help in determining vehicle type.
- Magnetometer
    - Magnetometers measure the magnetic field along one or more axes. This essentially translates into a digital compass. This is potentially useful for label predictions. I am less confident about this sensor's usefulness, but it will be easy to include and test its value.
- Gyroscope
    - The gyroscope gives data on the device's orientation and angular velocity. This is a sensor that is available on nearly all devices, and may provide some useful data. This is

the sensor I am least confident about, but this falls into the same category as the magnetometer: easy to include and assess its usefulness.

This results in the following list of data that I can collect and directly use, or infer from additional data points: timestamp, user ID, user location, user velocity, magnetic field, and device orientation.

From location and velocity, using a previous datapoint from one that is being measured, the following data can also be inferred:

- Change in longitude
- Change in latitude
- Change in velocity
- Geolocational shift- the combined change in longitude and latitude

What this means for the data collection app, is that I will be collecting all of this data, as well as the previously discussed metrics. As long as I am unsure of the usefulness of the magnetometer and gyroscope, I will implement their collection in the data collector app. Collecting more metrics during development is better than collecting less, as it will be easy to strip less useful metrics later on, whereas adding accurate- and entirely new sensor metrics for previously recorded datapoints is functionally impossible.

Once an initial version of the classification algorithm has been built, a later research question will aim to compare sensor data, and data that is considered useless will be stripped from the data collection in the final on-device module.

2. How can we solve the user privacy concerns around collecting data?

This project aims to collect user data, which means that privacy concerns may apply. In this chapter, I will look at what needs to be done to understand- and mitigate these privacy concerns.

Process and Solution

There are two main privacy regulations that the project needs to follow, namely the GDPR, which contains rules for the EU, and the AVG, which contains rules for the Netherlands specifically. Since there is no difference in privacy rulings between the GDPR and AVG, I will be looking at the AVG documentation.

Following the AVG, it would be easiest if the on-device module could avoid collecting personal data at all. This means that the individual providing the data we collect cannot be identifiable, directly, or indirectly. If the data that the on-device module collects can be anonymized, there would not be an issue beyond asking the user's consent to collect sensor data.

The stored datapoints consist of the following data. From my understanding of the AVG, among this data, some parts could be problematic and/or uncertain.

- User Identifier
    - While not directly linkable to a natural person since the on-device module collects no directly identifiable data, the user identifier will be unique to a user, and may be considered personal data.
- Timestamp
    - A timestamp on its own is not personal data.
- User Location
    - Since the GPS data details the exact location of a natural person, especially when linked with the timestamp, this might be a privacy concern and will need to be researched.
- User Velocity
    - This appears to be non-personal data.

The most problematic data that the on-device module aims to collect are the user identifier and the location data, and the fact that this data is linked as a data point. While these are not directly identifiably linked to an individual, the user identifier is unique to a device, or person, and location data also does not seem straightforward.

Whether what we process is considered personal data depends on the exact definition of 'personal data'. To find out what the law says about this, I have consulted the 'Handleiding Algemene verordening gegevensbescherming' by the Rijksoverheid. Chapter 3.2, of which the relevant excerpts can be found in Appendix 1, deals with this definition of 'personal data', and is what I will ground the definition on.

Following these rulings, GPS data is not inherently personal data. It however becomes personal data as soon as it is combined with other identifiable information, such as the unique user/device ID used in my datapoints. This means that certain actions need to be taken in order for the on-device module to be allowed to process this data.

The additional points are:

1. The on-device module needs to confirm user consent.
2. The on-device module needs to inform users what types of data will be collected, what it will be used for, how long it will be kept, and whether it will be passed to a third party.
3. The on-device module needs to only collect data that directly contributes to the stated goal of this module.
4. The data should be anonymized where possible.
5. Users need to be able to:
    a) Revoke consent at any time
    b) Receive a detailed overview of their collected personal data
    c) Request for their personal data to be destroyed
6. All personal data needs to be protected, which means that:
    a) Personal data needs to be securely stored
    b) Personal data needs to be protected against loss, destruction, deletion, and unauthorized access.
7. A record needs to be kept, detailing access times and details, and processing records
8. Access to the data by the organization collecting it needs to be restricted

Since this is quite a list, anonymizing data and avoiding the entire privacy problem might sound like the best solution. This is, however, not currently possible since a user's specific data history needs to be distinguishable in order to reconstruct every user's traveled routes; a function that is part of the route building algorithm. This means that this list of security extra measures will need to be implemented.

Concerning as this may be, for the purpose of rapid prototype development, I will only seek to implement GDPR compliance in the prototypes that directly interface with users/testers, or are otherwise hosted online. The data collection app, for example, is one of these, as is the database itself.

3. What is Flutter and how can it be used to build a mobile application?

Since the first app that aims to implement the on-device module once it is built is Baseflow's Gaon app, I will have to develop the on-device module to be implementable by the Flutter framework, as per the requirements set by the project owner. Since I start this project with zero knowledge of this framework, I will need to spend some time learning how it works. This is perhaps more of a personal development step rather than a research question, but I have included it in this report regardless.

Tutorials

I first watched some overview videos on YouTube, which generally explained the structure of flutter, which is also found in the official documentation[3], which I reviewed after. I also followed a couple coding tutorials[4] to get an idea of how build flutter apps.

From learning to set up the project, to managing flutter packages, to adjusting OS-specific configurations, and the flutter app framework itself, I believe to have built a sufficient understanding of the Flutter framework, and I can continue on to build prototypes for the next research steps, and later on the final products themselves.

Implementation

My main implementation of the flutter framework for this entire project is in the form of the data collector app. This app builds a datapoint with sensor data every 5 seconds that the functionality is active. The first prototype, however, was built to find a way to collect and interpret GPS data.

The image on the next page shows the code block for the main screen of this app. Flutter's structure is essentially a tree of widgets, where each element such as the 'Scaffold', 'AppBar', 'Text', 'Column', etc. is a widget that is passed to its parent widget. Going up this tree, it eventually ends up at the first widget that the app starts with, as shown in Figure 3.
This structure is fairly easy to understand, and each custom-made widget can include its own complex functionality. The app that the code in Figure 3 and 4 describes is a simple implementation of the Geolocator plugin, and displays the location when the button is pressed as text in the middle of the screen. This is the proof of concept that is also used to answer next research question. It uses an async

---

[3] *Flutter architectural overview*. (n.d.-b). Flutter. https://docs.flutter.dev/resources/architectural-overview

[4] Fireship. (2021, November 16). *Flutter Basic Training - 12 Minute Bootcamp* [Video]. YouTube. https://www.youtube.com/watch?v=1xipg02Wu8s

,Flutter Mapp. (2022, June 9). *Flutter Tutorial For Beginners In 1 Hour* [Video]. YouTube. https://www.youtube.com/watch?v=C-fKAzdTrLU

, freeCodeCamp.org. (2022, February 24). *Flutter Course for Beginners – 37-hour Cross Platform App Development Tutorial* [Video]. YouTube. https://www.youtube.com/watch?v=VPvVD8t02U8

method to fetch the current GPS position using the Geolocator plugin, which it then displays on the screen. A button is displayed on the home page that triggers this action.

```dart
void main() {
  runApp(const GPSApp());
}

class GPSApp extends StatelessWidget {
  const GPSApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ), // ThemeData
      home: HomePage(),
    ); // MaterialApp
  }
}
```

Figure 3: Basic flutter app code structure

```dart
class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  Position? _currentPosition;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Location"),
      ), // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            if (_currentPosition != null)
              Text(
                  "Position: ${_currentPosition}"), // Text
            TextButton(
              child: Text("Get location"),
              onPressed: () {
                _getCurrentLocation();
              },
            ), // TextButton
          ], // <Widget>[]
        ), // Column
      ), // Center
    ); // Scaffold
  }


  _getCurrentLocation() async {
    Geolocator.getCurrentPosition(
            desiredAccuracy: LocationAccuracy.best,
            forceAndroidLocationManager: true)
        .then((Position position) {
      setState(() {
        _currentPosition = position;
      });
    }).catchError((e) {
      print(e);
    });
  }
}
```

Figure 4: Geolocation proof of concept main page code

4. What does raw GPS data look like, and how can we refine it?

GPS data is one of the cornerstones of this project. It is therefore important to understand how to collect it. I am worried that this data collection step might be complex in some unexpected way, which is why this has been put forward as a separate research step.

Process and Solution

To understand how to work with GPS data, I built a small prototype flutter app that uses the Geolocator package[5], published by Baseflow, to collect GPS data.

Collecting GPS data ended up being more straightforward than I initially feared, and after going through the learning and troubleshooting in order to get the prototype running on an Android emulator, the resulting data is easy to work with.

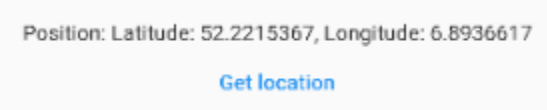

Figure 5: Readout of the geolocator prototype

The geolocator has methods to return the last known location, current location, as well as continuous location updates. These methods return a Position object, containing a longitude and a latitude. This is more straightforward than I expected; these values can be directly used for data processing.

The main question that remains is the frequency of data collection. For the development of the data collection app, this frequency will be arbitrarily decided to start with, and deciding on the final interval will be done in a later research step. The idea is to minimize this interval while keeping the same data quality, but it is not worth the effort to calculate- or predict this ideal interval now when I can simply use the data collected and drop a percentage to simulate longer intervals and get an accurate result that way.

Addendum

During development of the data collection application, I discovered that the Geolocator package was insufficient in its features. I needed the ability to record GPS data when the app was not actively in use, which was not available in this package. I therefore had to turn to a much more complex package[6] that did do this collection in the background. In order to implement this, however, I needed to write concurrency and parallelism into the application, which increased the complexity of this particular data collection significantly.

---

[5] *geolocator | Flutter Package*. (n.d.). Dart Packages. https://pub.dev/packages/geolocator

[6] *background_locator_2 | Flutter Package*. (n.d.). Dart Packages. https://pub.dev/packages/background_locator_2

Parallelism and concurrency are two sides of the same coin, both aiming to help solve the single-threaded nature of Dart, which is the programming language that Flutter is written in. Where concurrency is most often implemented by tasks using the asynchronous 'await' keyword, which waits for completion of the async task before continuing, parallelism works by creating and using Isolates. An isolate is essentially just a wrapper around a thread, and Flutter already runs in an isolate by default. This means spawning a new thread with its own memory running its own event loop. The reason why this is needed instead of a simpler asynchronous function is because of the previously mentioned running of the geolocation in the background. I am not fully aware of all the technical details of how the phone OS's handles suspended active apps, but from what I understand, the OS will terminate an app's base thread after it is inactive for a while, and flutter Isolates can be set up for background execution using callbacks.

First, this isolate is registered on an open port using a unique name in order to allow for communication between the isolates:

```
    IsolateNameServer.registerPortWithName(
        port.sendPort, LocationServiceRepository.isolateName);
    port.listen(
      (dynamic data) async {
        await backgroundPositionUpdate(data);
      },
    );
```

Figure 6: Isolate registration

The contents of the isolate are mostly controlled by the background locator plugin, and what I still have to do I register a callback function. There are a lot of configuration options for this, but the important part is that the BackgroundLocator's LocationUpdate is registered:

```
Future<void> _startLocator() async {
    return await BackgroundLocator.registerLocationUpdate(
        LocationCallbackHandler.callback,
        initCallback: LocationCallbackHandler.initCallback,
        disposeCallback: LocationCallbackHandler.disposeCallback,
        androidSettings: const bg_geolib_android_settings.AndroidSettings(
            accuracy: bg_geolib.LocationAccuracy.NAVIGATION,
            interval: 5,
            distanceFilter: 0,
            client: LocationClient.google,
            androidNotificationSettings: AndroidNotificationSettings(
                notificationTapCallback:
                    LocationCallbackHandler.notificationCallback)));
  }
```

Figure 7: Setting the background locator with specific parameters

This essentially handles the initialization and disposing of the callback, while causing the callback to trigger every 5 seconds, sending a locationDTO object (all the GPS related data) back to the main isolate

using the same port I set up earlier. There is more to this implementation than just the code blocks in the chapter; they have mostly been added to underline my explanation.

```
Future<void> callback(LocationDto locationDto) async {
  await setLogPosition(_count, locationDto);
  final SendPort? send = IsolateNameServer.lookupPortByName(isolateName);
  send?.send(locationDto.toJson());
  _count++;
}
```

Figure 8: Callback sending messages between isolates

Luckily, the accelerometer, magnetometer and gyroscope needed no such implementation. One unexpected benefit of this implementation is the addition of some new metrics that this package calculates from GPS data. This includes: speed, accuracy of this speed metric, accuracy of the longitude & latitude, and heading. These have mostly been added to the collected datapoints. I had already made a function that calculated a user's speed based on their difference in location and the timestamp, but since the package already included this, I removed it from my own implementation.

After understanding how to collect this data from GPS and sensors, I could move on to building the data-collection app.

Phase 3: Data Collection

The goal of this phase was to make an app that could be privately shared among a small number of people, in order to collect the varied data that the classification algorithm needs to train itself on.

In theory, this meant an app with a singular function. Something quite simple. However, in practice, it ended up several times more complex. In this chapter, I will describe the process of building this app, the challenges I encountered, and how they were dealt with.

App development: Initial version

The first version was fairly straightforward. Built around a mock-up of screen designs I made, as found in Figure 10, the primary functionality is simple. A user would pick a mode of transportation, and press start/stop. For the required GDPR compliance, I also needed to make functions that let the user delete/view their collected datapoints.

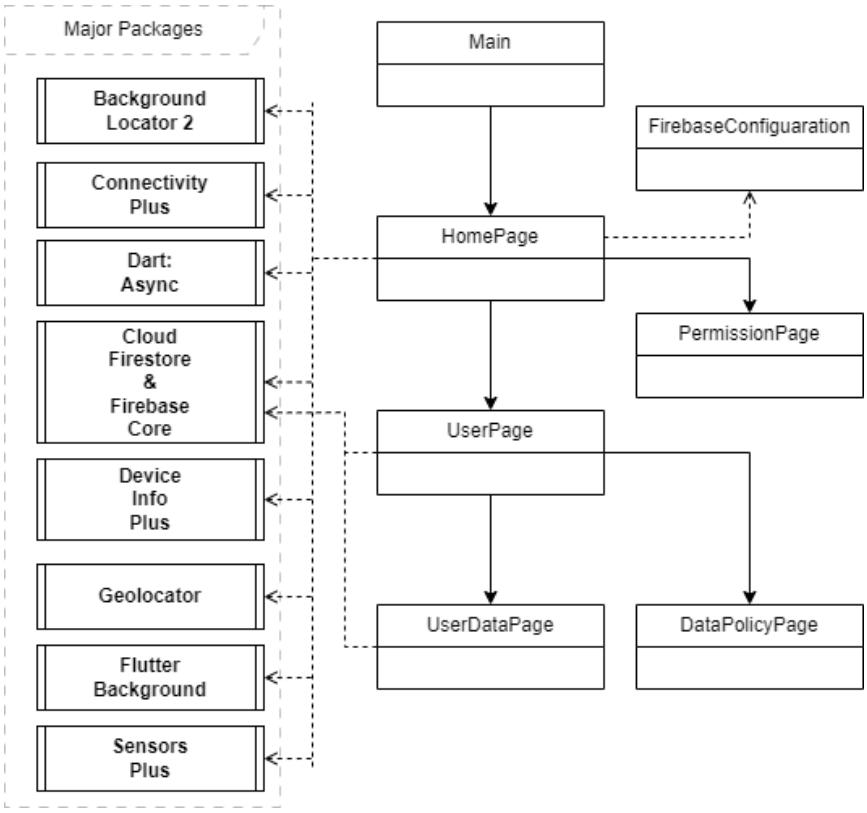

Figure 9: Simplified class diagram of the final app version

The trickier parts were implementing the data collection to run at set intervals, and setting up/communicating with a firebase database. Another part of complexity that came with this was the availability of an internet connection: I wanted to build a workaround to the possible problem of a spotty internet connection. Without an implemented solution, it would mean that if there was no internet connection, sending a datapoint would fail. Stopping the collection when no network was available would be easy, but not my preferred solution. Instead, I built a method that checks the connectivity status during the regular data collection moments, and if it ever returns a disconnect, it would start saving datapoints locally on the device, only to send them once a network connection was re-established.

I thought that the app was finished, and I could move on. However, that was not the case, and things quickly got more complex.

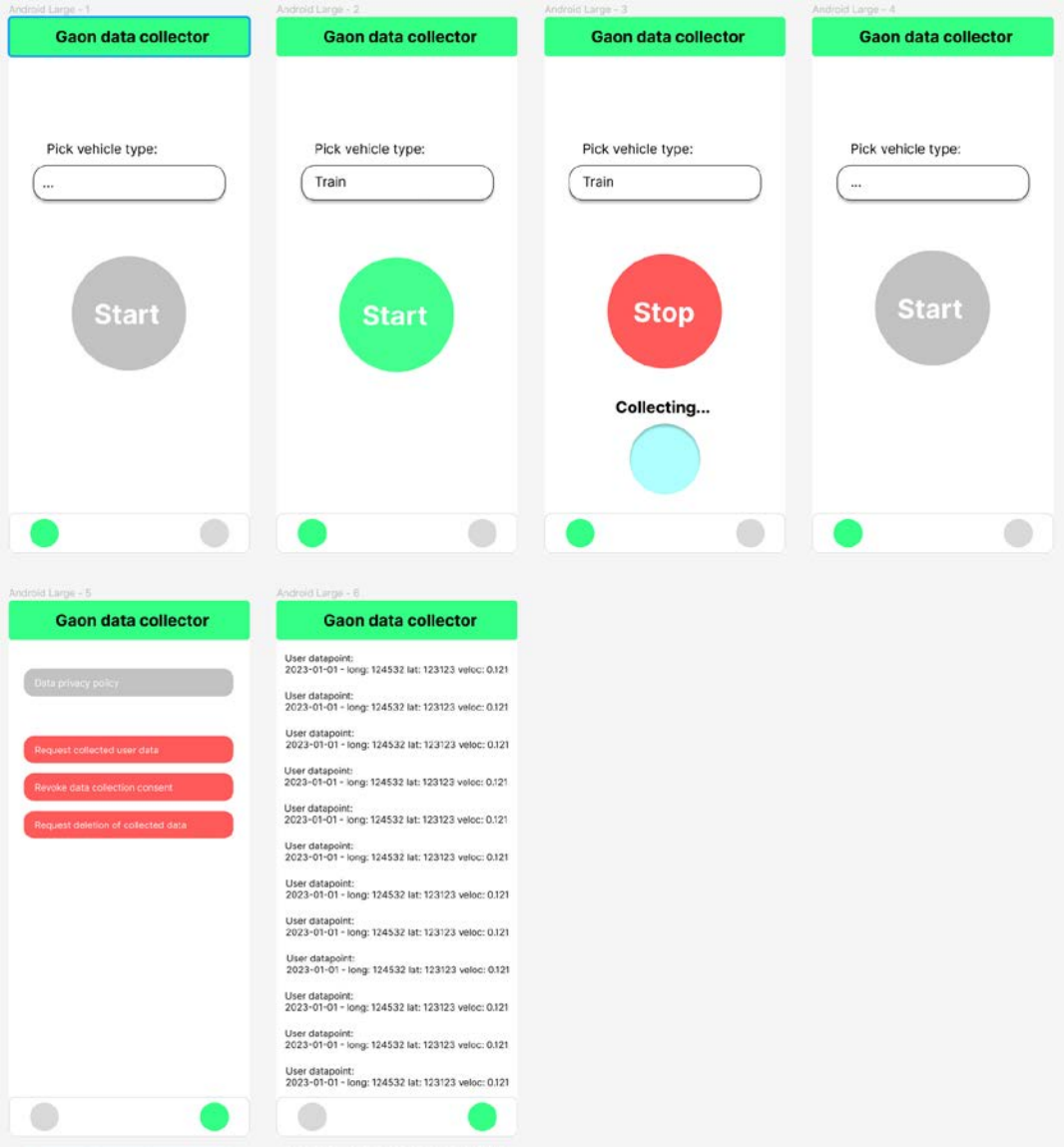

Figure 10: User Interface mock-up of the data collection app

Change in package and additional requirements

The first version had flaws. The biggest flaw being that data collection of the GPS was not occurring when the app was not in focus. What this ultimately led to was a redesign of the app, using a different package for GPS data collection which needed to run on a separate thread. This also meant building multithreaded communication, although the package had a decent guide to follow on how to do this. This implementation is also further explained in the previous chapter on GPS research.

Another challenge appeared because of this implementation. In order to properly run the GPS data collection when the app was not in focus, several additional user permissions were needed. Because if these permissions were not found, the functionality would not trigger. Beyond having to edit the android manifest for these app permissions, I also had to create several checks for the permission status, as well as request permissions from users, as that cannot be automated.

In general, I believe that permission popups should be implemented in a slightly user-friendly way, so I made another screen where these permissions were checked. This screen would stay up until all permissions were granted, and informed users which permissions were missing. Additionally, I set a check to run on a timer on the main page, so that these permissions would be constantly checked and requested if necessary, since they are needed for the app to function.

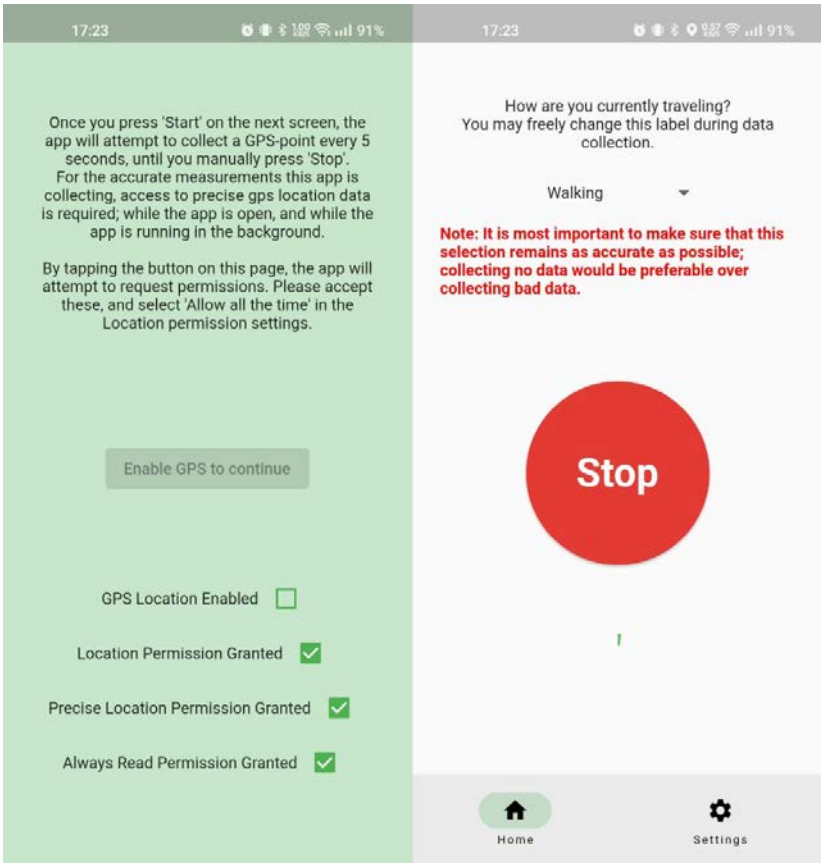

Figure 11: Permissions page and Home page UI

A final challenge has been working around the null safety requirement for Flutter. Variables should never reference null, or the app will cause a runtime error. This is a good feature because it shields you from unsafe code, but it also makes it tricky to work with asynchronous data. A lot of variables that I process in the app are values that are undefined at initialization. The accelerometer variable, for example, has no value until the app is initialized. For this, and many more, I had to use a combination of late initialization and assigning variables as possibly null, as seen in Figure 12.

```
late final FirebaseFirestore _db; // This avoids a runtime error by essentially promising that this value is initialized later. In this case, the firestore database connection is set up when the page is initialized.
num? _previousTimeStamp; //This avoids a runtime error by allowing the value to be null. On boot, there is no previous timestamp, this gets filled in later. In the code, I can use _previousTimeStamp! to indicate that I am sure that the variable has a value.
```

Figure 12: Examples and explanations of null safety implementations

The real complexity of this app comes not from the individual functions, but from the large number of packages and functions having to properly work together without causing issues, which makes it tricky to show. Each individual function is relatively small and easy to explain, but together the app consists of roughly 1500 lines of code. The class diagram from Figure 9 aims to show a simplified overview of the app. Some of the more interesting code fragments are listed below.

```
String fileContents = await FileManager.readLogFile();
String collection = kReleaseMode ? 'live' : 'dev';
firebaseCollection ??= _db.collection(collection);

if (_connectionStatus == ConnectivityResult.wifi ||
    _connectionStatus == ConnectivityResult.mobile ||
    _connectionStatus == ConnectivityResult.ethernet) {
  if (fileContents != '' && fileContents != null) {
    List data = json.decode(fileContents);
    for (final oldPoint in data) {
      firebaseCollection.add(oldPoint);
    }
    FileManager.clearLogFile();
  }
  if (dataPoint['distance_delta'] != null) {
    firebaseCollection.add(dataPoint);
  }
} else {
  if (dataPoint['distance_delta'] != null) {
    if (fileContents == '' || fileContents == null) {
      FileManager.writeToLogFileOverride('[${json.encode(dataPoint)}]');
    } else {
      List data = json.decode(fileContents);
      data.add(dataPoint);
      FileManager.writeToLogFileOverride(json.encode(data));
    }
  }
}
_previousTimeStamp = timeStamp;
_previousPosition = position;
```

Figure 13: Sending data to the Firestore database

The code in Figure 13 is the code responsible for sending the data to the backend. While this is not too special because setting up the firebase connection happens in another function, what makes this special is that it will dynamically adjust based on the user's connection status. It will directly send a point when an internet connection exists, but otherwise stores datapoints in a local file until a connection is found again, at which point it sends all the stored datapoints at once.

Before this code, the datapoint has already been assembled. This piece of code is strictly about sending

the data. First, it reads the file containing potential points collected while offline, and then attempts to connect to the database. If a connection is found, it will send datapoints found in the offline local file, if any, and add the newly recorded datapoint as well. If there is no connection available, it will simply append the current point to the offline local file.

```
void _checkPermissions() async {
  bool serviceEnabled = await Geolocator.isLocationServiceEnabled();
  LocationPermission perms = await Geolocator.checkPermission();
  if (perms != LocationPermission.always || !serviceEnabled) {
    if (_permissionTimer!.isActive) {
      _permissionTimer!.cancel();
    }
    _goToPermissionsPage();
  }
}
void _goToPermissionsPage() {
  if (mounted) {
    Navigator.push(context,
            MaterialPageRoute(builder: (context) => PermissionPage()))
        .then((value) => {
              initializeFlutterBackground(),
              _permissionTimer = Timer.periodic(const Duration(seconds: 2),
                  (Timer t) => _checkPermissions())
            });
  }
}
```

Figure 14: Permission checking

The two functions in Figure 14 are part of another functionality of the app. On startup, the app checks if location permissions are granted and the GPS functionality is enabled using the _checkPermissions() function. If they are not, another page will open and request permissions. It will then only redirect users back to the home page once all permission checks pass. Once back on the home page, besides checking all of these permissions before attempting to record a datapoint, this check is also initialized on a 2-second timer.

Rollout and Data exploration

One of the requirements from the project supervisor was a rollout to testers via the Play store. Using an internally shared APK would have been easier, but I am happy for the experience of using the dev console on the Play store in order to release an app for internal testing. I had to make sure my app was following certain guidelines as set out by google, and add some explanations on what I was using the collected GPS data for.

After making the data collection app and recording a test run, I collected a dataset of roughly 700 points over roughly one hour, minus a few minutes where collection failed, roughly between 19:20 and 19:30. This dataset was recorded largely on a train, with a short walk afterwards. After writing an application to download and process this data, I visualized it in a few graphs in order to see which metrics might be valuable.

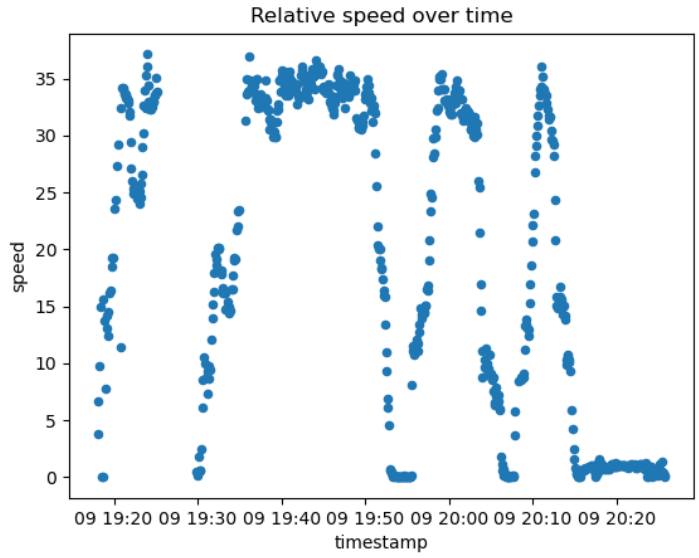

Figure 15: Relative speed over time of a single journey by train and foot

This first graph shows a clear distinction in speed between train and walking. However, this also shows that simply using only speed as a classification metric would result in inaccuracy, since a train stands still at train stations. The pattern for the train is quite consistent however, in it's top speed, and its signature peaks and valleys where the train speeds up and slows down over a longer duration.

Zooming in on the speed graph for the part that was recorded on foot gives the following graph.

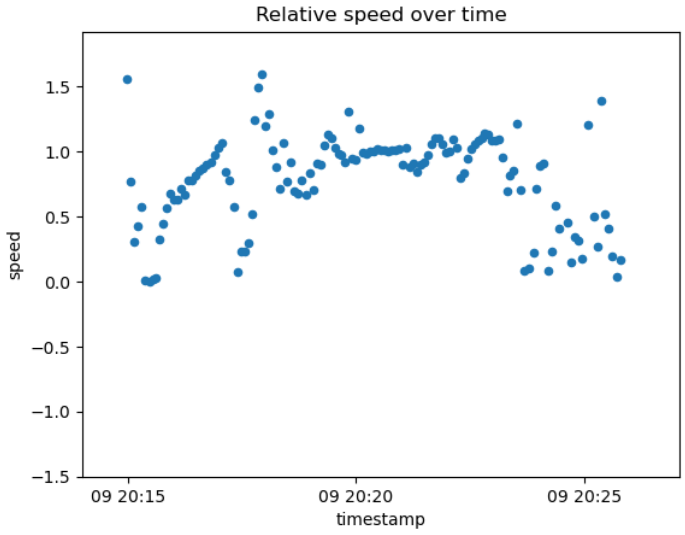

Figure 16: Relative speed over time of walking datapoints

This chart shows how walking will result in a steadier pattern, but with quite some points approaching zero, where the user can be assumed to be standing still. Still, the general shape of this part is distinct when compared to the train.

The speed metric on its own will likely never be enough to truly make a distinction, due to the possible edge cases. This is where other metrics come in handy. Take the following graph for example:

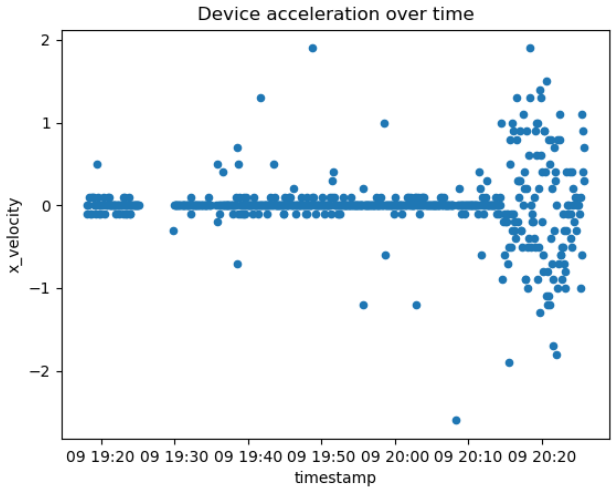

Figure 17: Device accelerometer data over time

This is the same dataset, however instead of speed, it compares one of the values the device's accelerometer is recording. This shows a distinct pattern between train travel and walking, and will help in making distinctions between types of transportation. Another type of data that supports this distinction, arguably even better, is the recorded accuracy of the GPS' calculated speed:

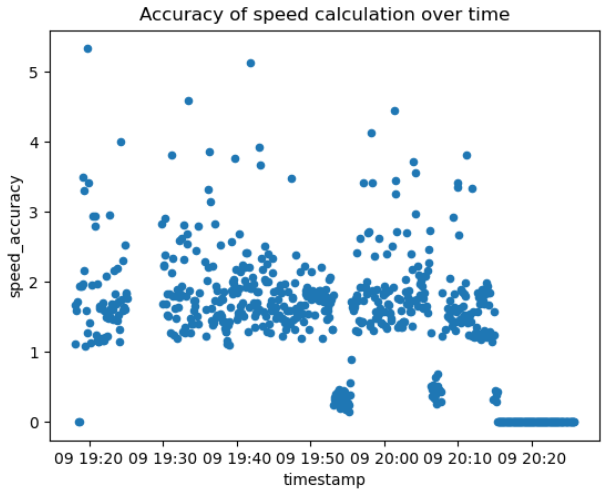

Figure 18: Rated accuracy of speed measurements over time

This seems to handle the distinction between train stations and walking better. Some additional data will have to be collected to discern whether this is because a train is technically a barrier between the GPS signal and the phone, while being outside and walking gives a direct link, or whether it is because of a

relatively higher signal noise in a busy train, or whether there is some other reason. If the results are consistent between these tests, this metric could be strong for classification.

After combining some days of collected data by employees, and looking at the data from a different perspective, namely by plotting the Label on the X-axis, other distinctions become clear. I have compiled some of these graphs below.

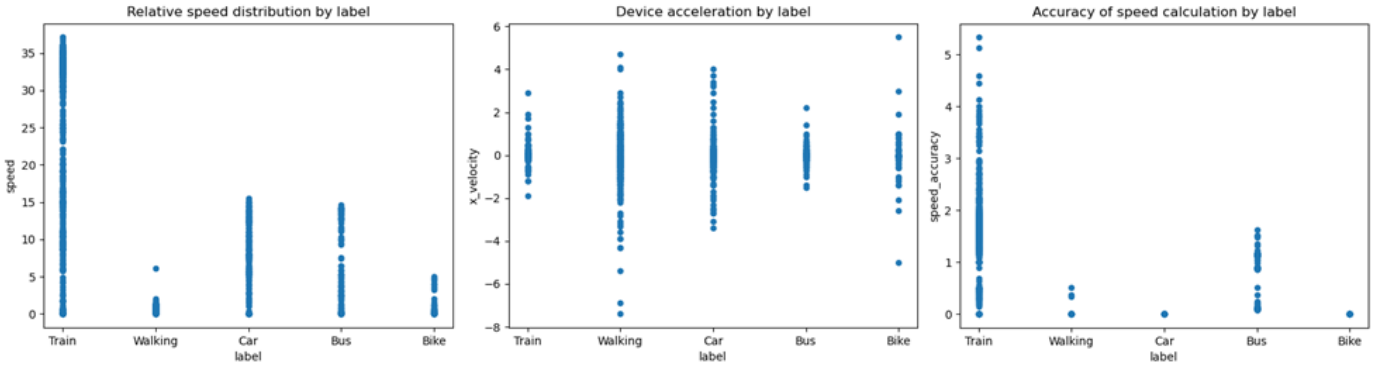

Figure 19: Graphs comparing several metrics against the specific recorded labels

These graphs show clear distinctions in speed distributions, as well as turbulence level in acceleration and accuracy by mode of transportation. This shows that these metrics have potential to be useful in classification.

This is still a fairly limited dataset, and more will need to be recorded over the coming weeks, but it is a good start. There are other recorded metrics, but they largely give the same information as the ones discussed in this chapter. The main question of whether this data alone is enough remains. Metadata may be another angle to research, as any additional good metadata may provide a significant increase in accuracy.

Additional Sensors

After seeing this data, I decided that I should leverage every easily available sensor I could access on most android devices, in order to hopefully expand the depth of my dataset. For this, I had to rework the app slightly. I also took this opportunity to take a look at the database structure. Firestore is a document storage system, where collections contain documents that each can be completely different, and each document can contain subcollections. I spent some time seeing if I could optimize my dataset by adding subcollections, but eventually decided against it when I found that I could directly query documents without using additional billable Firestore capacity for other documents in the collection.

All in all, this took a great deal more time than I initially intended to spend on this part. A silver lining being that much of the code can be reused for the eventual on-device module. I might've been able to get away with dropping or simplifying some of the functionality, but I did not, because I wanted to release a polished app. While I am happy with what I made, I do think, in hindsight, that I should have cut some corners.

Phase 4: Additional Research

5. Is it possible to employ machine- or deep learning techniques in user vehicle classification, and if yes, how?

Description

In order to have the most accurate classification possible, I need to look at the technologies that I know of. Additionally, these machine learning subjects have been part of my study's specialization, and adding them to this project would increase its complexity. While this benefits me since it allows me to show my ability to build such systems, it is also beneficial to the application I aim to build, since machine learning is a powerful tool that can efficiently aid in classifying modes of transportation.

The research question is therefore slightly misworded. I know that machine learning can be used in this process. I am unsure if deep learning can be used. While I think that deep learning can be a superior method, I only experienced using it on image-based datasets, so part of this research step is finding out if it can be used for numerical data. This means that to answer this question, some sub questions need to be answered:

1. Can deep learning be used to classify numerical data?
    a. Will it be beneficial to use deep learning over a traditional machine learning classifier?
    b. If not, can I create metadata in the form of an image, in order to aid in classification via deep learning?
2. How is the quality of the data I currently collected?
3. Can I construct additional metadata metrics that are not correlated with existing metrics but do provide additional value?
4. Will making a machine learning algorithm based on this data achieve a sufficiently high confidence score so that I can implement it in the final solution?

Process and Solution

Can deep learning be used to classify numerical data

So far, I have used deep learning only with image-based data. The answer to this question, however, was very straightforward. Through some quick google searches [7]I have found that deep learning can be used with numerical data, but it may not work as well as a decision tree or random forest classifier when working with smaller datasets.

[7] Ronaghan, S. (2018, August 7). Machine Learning: Trying to predict a numerical value. *Medium*. https://srnghn.medium.com/machine-learning-trying-to-predict-a-numerical-value-8aafb9ad4d36

Classify structured data with feature columns. (n.d.). *TensorFlow*. https://www.tensorflow.org/tutorials/structured_data/feature_columns

6. Which traditional machine-learning algorithm gives the best result for my dataset

My next step, then, is to create some machine learning models and feed them the collected data so far, in order to find the best implementation. In order to do this properly, I will research and discuss some machine learning models/strategies that may fit this project's use case.

A starting point for picking the right machine learning algorithm would be the scikit-learn algorithm cheat-sheet. Scikit-learn is very commonly used library of tools, packages and guides centred around machine learning and data analysis. Their solutions are used in apps like Spotify, J.P.Morgan, and Booking.com, as well as many other machine-learning applications.

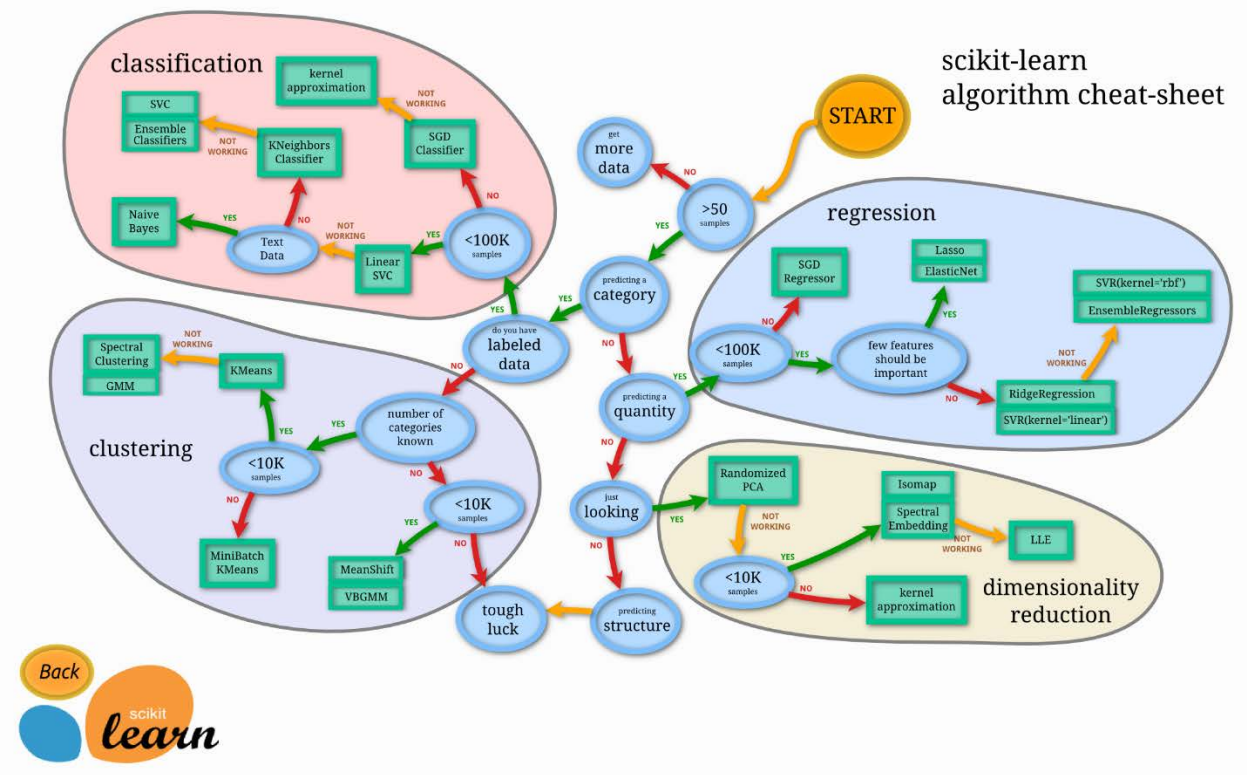

Figure 20: Scikit-learn algorithm cheat-sheet

This diagram gives a solid overview of some of the most used machine learning algorithm, with a flow of how to get there. In order to properly pick the best one, the data that I have needs to be considered.

The data collected for this project:

- Currently has <1K samples. This is expected to increase over time.
- Has known categories. For the scope of this project, no desire exists to differentiate between exotic modes of transportation such as shuttle buses, loan bikes, or carpooling.
- Is labelled. **However,** the final on-device module will not be collecting labelled data. This means that the initial set of labelled data will only increase in volume if more people use the internal data collector app. Consequently, once the project is over and the on-device module implemented, the amount of unlabelled data will increase at a much greater pace than labelled

data, if the amount of labelled data increases at all. Since I want to implement continuous training, this is an important distinction.

- Is numerical.

When following the chart, this gives us either the Linear SVC classification algorithm, or the KMeans clustering algorithm, as the theoretically best picks.

EC3

Using either of these algorithms would mean compromising on the input data, something I wished to avoid. I instead did some research on this topic, with the main question being: 'What if there was a way to combine both classification and clustering, using the strengths of both, in order to arrive at an altogether superior algorithm?' This is where I found EC3. EC3, or it's variant for more imbalanced datasets like the one used for this project, iEC3, is exactly what I was looking for. I cannot explain the algorithm better than its creator, so I have included some excerpts from the abstract below[8].

"Classification and clustering algorithms have been proved to be successful individually in different contexts. Both of them have their own advantages and limitations. For instance, although classification algorithms are more powerful than clustering methods in predicting class labels of objects, they do not perform well when there is a lack of sufficient manually labeled reliable data. On the other hand, although clustering algorithms do not produce label information for objects, they provide supplementary constraints (e.g., if two objects are clustered together, it is more likely that the same label is assigned to both of them) that one can leverage for label prediction of a set of unknown objects. Therefore, systematic utilization of both these types of algorithms together can lead to better prediction performance…
…We additionally propose iEC3, a variant of EC3 that handles imbalanced dataset…
…We show that our methods outperform other baselines for every single dataset, achieving at most 10% higher AUC. Moreover our methods are faster (1.21 times faster than the best heterogeneous baseline), more resilient to noise and class imbalance than the best baseline method."[9] (p. 1)

In other words; using a model that implements both clustering and classification should result in more accurate classification, while supporting imbalanced datasets.
For a traditional Machine-Learning approach, this seems to tick all the boxes of my requirements. The main question is if this algorithm can be implemented by me, or if it is too complex to do so. I have found a single GitHub repository with some example code[10].
Sadly, I could not find an easy way to implement this into my prototype, since the Flutter library I use to

[8],
[9] *Chakraborty, T. (2017). EC3: Combining Clustering and Classification for Ensemble Learning. Journal of Latex class files, 13(9), doi:10.48850*
[10] Lcs2-Iiitd. (n.d.). *GitHub - LCS2-IIITD/EC3-Combining-Clustering-and-Classification-for-Ensemble-Learning: EC3 is based on a principled combination of multiple classification and multiple clustering methods using an optimization function.* GitHub. https://github.com/LCS2-IIITD/EC3-Combining-Clustering-and-Classification-for-Ensemble-Learning

run the classification algorithm on-device does not support custom models, and I could not find an alternative. If the final model turns out to be insufficiently accurate, I may attempt to manually write an implementation using this example code, or simply leave it as a recommendation for future development.

Deep Learning with TensorFlow

From my experience in deep-learning, it is overall easier to set up, and generally performs on-par or better than traditional machine-learning algorithms. The primary question I need to answer is whether or not my data fits with the possible input for deep learning. I've mentioned before that I've only done image-based deep learning thus far, and I need to find a method to incorporate the sets of numerical features I've collected into deep learning.

Incorporating image-based deep learning

One idea I have kept considering from the start, is to use the longitude and latitude present in all GPS datapoints to request a closeup image of the directly surrounding map data of those coordinates from the Google Maps API. Each vehicle type is more likely to be found on certain roads; a biker in a bike lane, or busses and cars on open roads. This, in theory, should mean that having an image of the road a user is traveling on, would give the algorithm a way to classify them to a degree. To do this, deep learning can be used.

Deep learning image recognition works by training a neural network to recognize patterns, which can be used in this scenario to recognize the types of roads a user is traveling on.

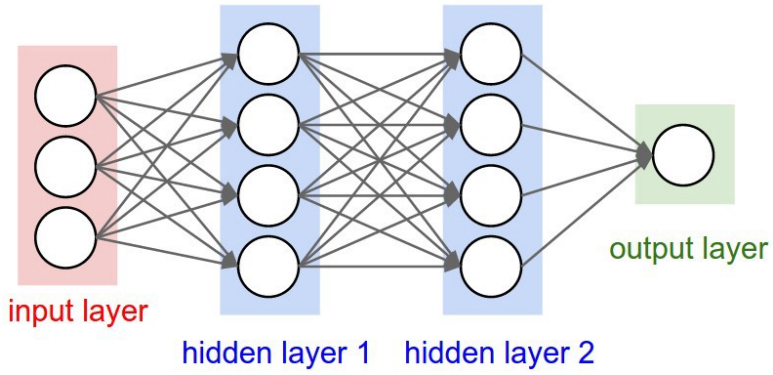

Figure 21: Overview of a neural network

This works by giving the neural network input data, which is then processed by one or more hidden layers, each of which contains a number of nodes that individually computes weights of features. Linking these together creates a complex 'neural network' that finally concludes in one output layer, which is the output of the classification. Image classification in this way is a powerful tool, but requires a large dataset and a relatively large amount of computational power.

This is an alternative method to my current classification based on numerical features. I would use this in conjunction with my machine-learning algorithm in order to hopefully make the final classification result more accurate, which may or may not end up being necessary depending on the accuracy of the regular machine-learning models. I briefly considered using a locally stored map, but this file would be much too large for an average user to consider installing. There are a couple concerns with this API-based approach, however, and I will consider these next.

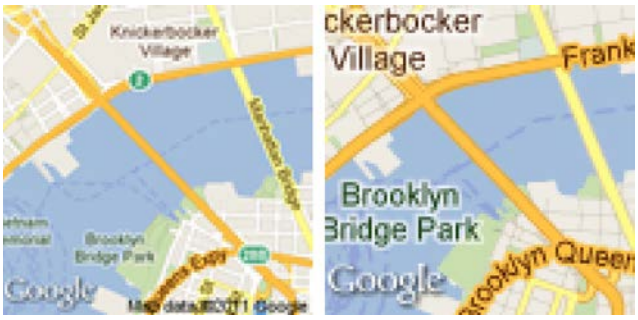

Figure 22: Comparison in detail between a 200x200 (left) and 100x100 (right) pixel image

a) The price of using the Google Maps API. The first 28.500 requests per month are free, after which it quickly starts costing a lot[11]. As for the expected usage; if I record a datapoint every 10 seconds, set the average trip duration as 45 minutes, and account for 500 trips per day to start, I would need 138.000 requests per day. The scale of this operation makes it difficult to recommend, if I were to use all data to train with.
   An alternative option to this would be to host a map on Baseflow servers. I have found OpenStreetMap to be a highly detailed, free to host alternative to google maps, and Baseflow has expressed interest in hosting this for more than just this project, so this may be an approach. One foreseeable issue is that this would go against on-edge calculation, since the model would need an active internet connection to query this hosted OpenStreetMap API. Having a map copy on device is not an option, since that would quickly scale required storage space.
b) Storage size. The pictures wouldn't need to be large. The main point of these would be to show the road type around the GPS coordinates. Comparing 100x100 pixels and 200x200 pixels, and calculating based on the 28.500 items as described in the previous point, 100x100px images would take up 171MB per month, or 2.1GB per year, and 200x200px images would take up 513MB per month, or 6.2GB per year. This sounds acceptable.
   Alternatively, when using the hosted OpenStreetMaps API, this storage cost could be entirely avoided by only storing the image of the current point in memory.

This seems acceptable, which means an image-based deep-learning approach is possible. It would output a classification score which I could either integrate with the traditional machine learning algorithm, or combine with, if the implementation ends up being possible, the numerical deep learning neural network. My hypothesis is that the composite of both a model trained on this map data, and a model trained on the numerical data would perform better than either one on its own.

[11] *Platform Pricing & API Costs - Google Maps Platform*. (n.d.). Google Maps Platform. https://mapsplatform.google.com/pricing/#pricing-grid

Maps Static API Usage and Billing. (n.d.). *Google for Developers*. https://developers.google.com/maps/documentation/maps-static/usage-and-billing

7. How can an accurate prediction of vehicle type be made?

The base result from building the machine learning algorithms is a classifier with a high rated accuracy. This is relatively unexpected, but the question remains whether or not this result is entirely trustworthy. I have some reservations in this regard since the data that has been recorded is based on mostly the same trajectories and the same devices. This is not something that is easily fixed before the project's deadline, but I can consider some options to mitigate this flaw.

One of the main concerns with predictions via machine-learning is that edge-cases may not accurately be classified. A datapoint recorded when standing still will, in terms of speed and acceleration, look the same for all modes of transportation. In order to increase the accuracy of edge cases, some solutions can be considered:

- The earlier proposed solution of fetching street-map data and using high resolution (as in small-scale/zoomed in) images of the local area around every datapoints' location offset by the recorded location's accuracy, and training a deep-learning model on them. This solution is likely powerful, especially when combined with the current classifier.
- Using previous datapoints to calculate meta-metrics. For example, if I can find a way to express the trend of user speed and acceleration over time as one or more variables, I can compensate for the volatility of the sensors. This approach has its own issues. It will lessen the chance that a change in transportation mode (e.g., getting of a bus and starting to walk) is properly detected, and it will be ineffective for the first few recorded datapoints.
- Adding labels for standing still. This alleviates the primary exception, but is not desired as per the project's requirements.
- Alternatively, for standing still, the on-device module can simply assign the previous label when recording datapoints with a speed below a certain threshold (to account for sensor volatility). This might just be the best solution for this problem, since we can assume that if someone isn't moving, they also haven't changed their mode of transportation in any meaningful way, and classification-based labelling can be resumed as soon as movement is detected.

A combination of these strategies can be implemented to increase the accuracy of labelling. Given the time-constraints of the project, some of them are more likely to be implemented than others, but they will still stand as advice for further development.

8. What is the minimum acceptable quality of collected data, and how can we optimize this for user's phone data usage?

For this question, it is first necessary to define data quality. In the context of this project, data quality can be described as ''the degree in which collected data leads to accurate classification results." In other words, higher quality data means that less overall data- or less frequent data would be necessary in order to reach a similarly accurate classification certainty. What this means for the data collection for the prototypes of this project, is that a collection of strong features/metrics need to be collected and interpreted in order to reach the highest classification score possible.

In order to find the impact of certain changes to the dataset, I have set up an experiment. Certain steps in creating and training the classifiers use random states. I have set the random states of all relevant steps in the classification process to a static value. This means that every time the classifier is ran, the resulting accuracy report will be the exact same. I can then change one variable each run, and note the impact. I am testing it with three different values (42, 0, and 1337) to avoid outlier results. I picked this number of values because I need to manually change them each time, and this seemed like a decent balance between speed and accuracy. For this entire experiment, only the highest-ranking classifier from the ones I have tested has been used.
The following table lists all the data changes I've made, and the impact they have on the overall accuracy of the classifier. I have listed the train-test-split accuracy score as well as the f1-score.

The average of original values are as follows: accuracy = 0.910, f1_score= 0.903

Table 8: Impact of certain changes to the dataset, expressed as accuracy and f1

Type of change from the original data	New accuracy	Difference	New f1-score	Difference
Removing speed below 0.15 (standing still)	0.908	-0.002	0.90	-0.003
Removing gyroscope values	0.912	+0.002	0.90	-0.003
Removing magnetometer values	0.926	+0.016	0.923	+0.020
Removing heading values	0.926	+0.016	0.92	+0.017
Removing accuracy values	0.912	+0.002	0.90	-0.003
Removing speed accuracy values	0.902	-0.008	0.893	-0.010
Changing data frequency to 1 per 10 sec	0.920	+0.010	0.913	+0.010
All of the above except frequency change	0.876	-0.034	0.89	-0.013
All of the above positive differences, except frequency change	0.9413	+0.031	0.937	+0.034
Removing speed below 0.15, gyroscope, magnetometer, and heading	0.9413	+0.031	0.94	+0.037

What this tells me is that individually removing one or two metrics does not significantly affect the quality of the classifier, but removing multiple has a compounding impact. Removing gyroscope, magnetometer, heading values, and all speed values below 0.15 seems to have a positive result. The reason why I included the <0.15 speed values to be removed is because this is a proposed solution to the standstill-labelling issue as discussed in the previous research question.

One last thing to keep in mind is that while these differences seem small, they are actually very meaningful in the context of remaining inaccuracy. This remaining inaccuracy is 9% in the original set, so a change from 9% inaccuracy to 5.87% inaccuracy is an improvement of 34%. The main issue with these numbers is that while I did account for some of the randomness by using the average of 3 random state values, I do not feel that this compensates properly for the fluctuation in scores caused by the random state changes. I would likely need to run several dozen- if not hundreds of random states in order to get a precise rating, but this will take more time than I would like to spend on this step.

Regardless, moving forward, I will be using the dataset without gyroscope, magnetometer and heading, and without the speed values below 0.15.

Is it possible- or desirable, to limit data collection to only occur when movement is detected?

One answer to this question comes from deliberation done in the previous chapter. One way to solve the issue that arises when movement is close to- but not zero, which it will almost never reach due to the GPS sensor's natural deviance, is to simply record the previous label as the current one. If someone is completely stationary, it is unlikely that they are actively changing their mode of transportation, and even if they were able to do this without moving at all, it is unlikely that any algorithm could correctly detect this change, since all variables would stay the same. Simply recording the known label, and resuming classification once movement is detected can be beneficial.
This would mean that no calculations- and therefore no data collection needs to take place in order to classify someone who is not moving.
The main argument for keeping data collection active while standing still is the consistency of the data. Making all datapoints have the same n-second-interval makes it much easier to find the cut-off-point for travelled routes and reconstruct these routes, so this is something that I want to keep.
However, collecting data inevitably leads to higher power consumption. Since one of the requirements of the on-device module is to limit this power consumption where possible, it means that this is one place where efficiency gains could be made.
In order to solve the consistency problem, I could, as discussed, simply leave out the computationally expensive classification step, while still recording the datapoint itself, using the previously recorded label until the user starts moving again.
In other words: Possible, yes. Desirable? No.

Phase 5: Realisation

Prototyping a Machine Learning Model

Predicting labels is the main goal of the on-device module. This means that a machine-learning model needs to be trained and implemented. This chapter will go over the work done to prepare, build, and train the classifier. This work has been done in Python, using the Scikit-learn library, which is a collection of tools for machine learning. Jupyter Notebook was used for the writing and execution of code.

In order to have a good dataset for machine learning, some preparation needs to be done. Since the dataset is constantly being updated, this step is done regularly. It involves downloading all the datapoints stored in the firebase database and combining them into a single dataset. A few steps are then taken.

Data clean-up and preparation

In order to have the best classification result, I need to make sure my data is as reliable as possible. I do this by manually removing sets of datapoints which I know are incorrectly labelled during data collection. Some human error occurs during data collection, like leaving the collection app on when someone has already finished their trip. After communicating this, I know which segments to remove, which I can do since I store the device id and a timestamp on each datapoint. This method may not catch all erroneous data, but the data collection team is small, and I trust them to tell me these errors.
An example of doing this is Figure 23, which is a python code fragment that connects to my database and deletes the most recent 5 minutes.

```
docs_ref = db.collection("live").order_by("timestamp", direction=firestore.Query.DE-
SCENDING).limit(60)

docs = docs_ref.get()

for doc in docs:
    doc.reference.delete()
```

Figure 23: Example code for deleting database datapoints automatically

In most datasets, removing or replacing null values would be required, but since I remove the null values on the device before recording a datapoint, there are no leftover null values. Another part of pre-processing is the removal of outliers. I am currently not doing this, since the data describes plenty lot of outliers, or edge cases, that need to be accounted for in the prediction.
I then create a correlation matrix in order to find weak and strong features, using a built-in function of pandas dataframes:

	speed	speed_accuracy	gyroscope_z	userAccelerometer_z	magnetometer_z	magnetometer_x	gyroscope_x	heading	magnetometer_y	userAccelerometer_x	userAccelerometer_y	accuracy	gyroscope_y
speed	1.000000	0.312091	-0.008315	0.077005	-0.236664	-0.085374	-0.078008	0.132721	0.190532	-0.022731	0.165555	-0.020143	0.008293
speed_accuracy	0.312091	1.000000	-0.024576	0.000865	-0.062459	-0.104915	-0.060162	-0.163971	0.134635	-0.035658	0.131360	0.007907	0.029869
gyroscope_z	-0.008315	-0.024576	1.000000	-0.143603	0.006336	-0.016559	0.100561	0.034488	0.014363	-0.073214	-0.050219	-0.001589	0.168095
userAccelerometer_z	0.077005	0.000865	-0.143603	1.000000	-0.042919	-0.037500	0.044819	-0.005568	-0.034376	-0.136308	0.026738	0.003287	-0.123249
magnetometer_z	-0.236664	-0.062459	0.006336	-0.042919	1.000000	0.368580	0.026826	0.074953	-0.074970	0.033591	-0.138647	0.109500	-0.046345
magnetometer_x	-0.085374	-0.104915	-0.016559	-0.037500	0.368580	1.000000	0.045421	0.144850	-0.042046	0.036241	-0.079660	0.034752	-0.022163
gyroscope_x	-0.078008	-0.060162	0.100561	0.044819	0.026826	0.045421	1.000000	-0.001695	0.049959	0.033033	-0.068452	0.001039	0.166576
heading	0.132721	-0.163971	0.034488	-0.005568	0.074953	0.144850	-0.001695	1.000000	-0.106557	0.018084	-0.062669	-0.082832	-0.009163
magnetometer_y	0.190532	0.134635	0.014363	-0.034376	-0.074970	-0.042046	0.049959	-0.106557	1.000000	0.037924	0.155684	0.070043	0.031025
userAccelerometer_x	-0.022731	-0.035658	-0.073214	-0.136308	0.033591	0.036241	0.033033	0.018084	0.037924	1.000000	-0.106740	0.003171	-0.141711
userAccelerometer_y	0.165555	0.131360	-0.050219	0.026738	-0.138647	-0.079660	-0.068452	-0.062669	0.155684	-0.106740	1.000000	0.023034	0.069415
accuracy	-0.020143	0.007907	-0.001589	0.003287	0.109500	0.034752	0.001039	-0.082832	0.070043	0.003171	0.023034	1.000000	0.011571
gyroscope_y	0.008293	0.029869	0.168095	-0.123249	-0.046345	-0.022163	0.166576	-0.009163	0.031025	-0.141711	0.069415	0.011571	1.000000

Figure 24: Correlation matrix of all recorded metrics

The correlation matrix in Figure 24 shows, as the name suggests, whether any of two features are highly correlated or not. This comparison produces a value. The closer this value is to 1 or -1, the more correlated the features are. Removing one of a pair of correlated values will improve the quality of the dataset. The main values that jump out here are the magnetometer X,Y, and Z values, and the speed – speed_accuracy pair. The correlation value is not too high, however, so I will let them be. I can, later on, test out the accuracy difference of the model by removing these features, and noting whether or not there is a significant difference in accuracy.

My final pre-processing step is a simple evaluation of feature strength. I do this by using a function called Gini Importance or Mean Decrease in Impurity. The Mean Decrease Impurity (MDI) is defined as the total decrease in node impurity (weighted by the probability of reaching that node (which is approximated by the proportion of samples reaching that node)) averaged over all trees of the ensemble[12]. This essentially means that a higher score means a feature that is better at differentiating between labels.

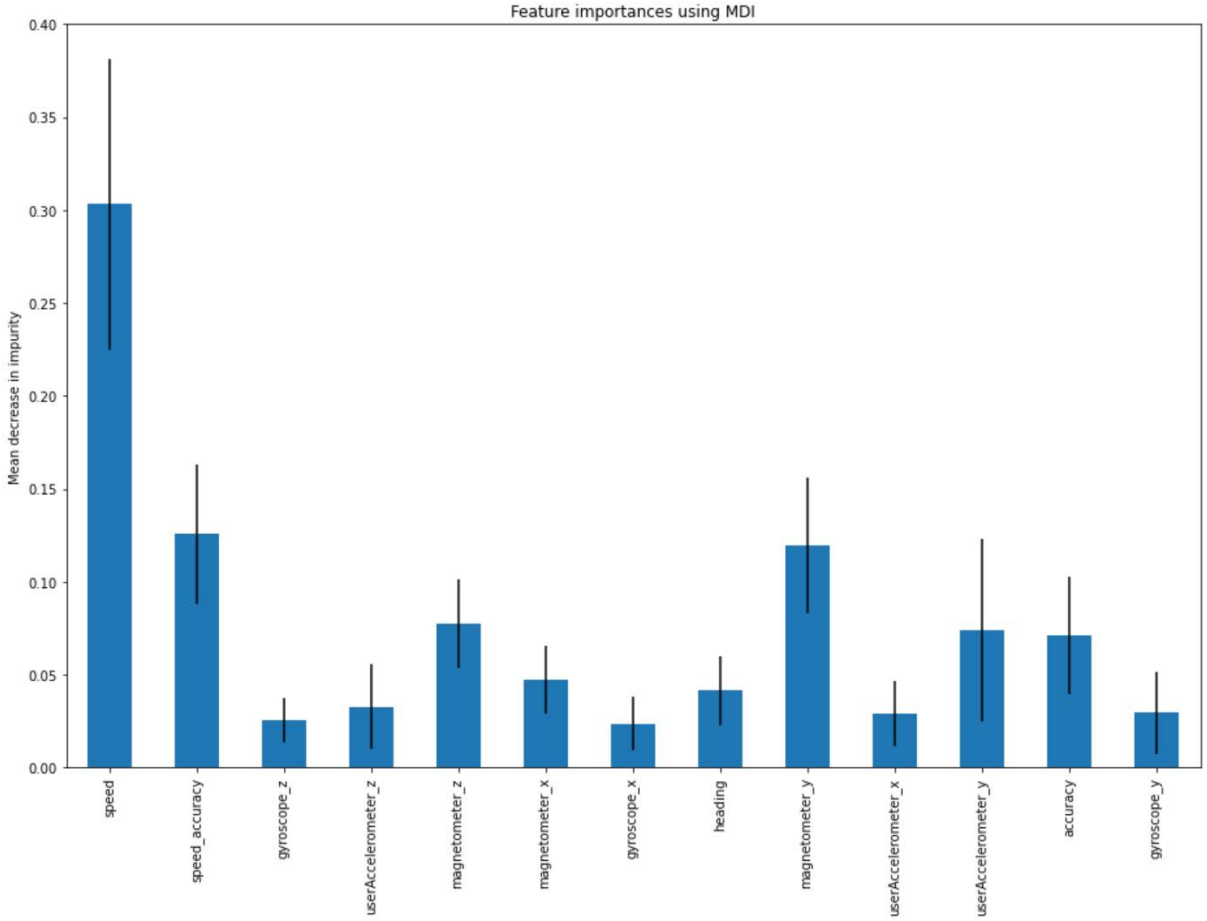

Figure 25: Graph of feature strengths based on the mean decrease of impurity strategy

[12] Lee, C. (2020, September 8). Feature Importance Measures for Tree Models — Part I. *Medium*. https://medium.com/the-artificial-impostor/feature-importance-measures-for-tree-models-part-i-47f187c1a2c3

While these values are not particularly high (each is rated between 0 and 1), it does give insight into the quality of each feature, where speed is by far the best feature. Surprisingly, the magnetometer is also a high scorer, while the gyroscope is relatively low. This means that it is likely that stripping the gyroscope from data collection would not impact the accuracy of the machine-learning model negatively, and might even improve the quality.

Model comparison, training and testing

My dataset is, sadly, slightly imbalanced. When one mode of transportation is significantly underrepresented, it may be absent from some segments of the training- or testing dataset. For earlier iterations, this means that I had to compensate for this. I did this by duplicating some of the datapoints. This is an inelegant solution, however, and after accumulating enough datapoints for this error to no longer occur, I removed this duplicate data. This resulted in a slightly lower accuracy score, but built from a more accurate dataset.

My next step was to manually train and compare different classification algorithms. I first picked SVC and K-Nearest-Neighbours, as they were theoretically the most appropriate for my data, and added to this list several classifiers, both ones I've used before, and some that I found online. The goal of this is to have a sizeable selection of classifiers, and to pick the best one I can reasonably find. In total, I used 7 classifiers. This turned out to be a good thing, since the best classifier was not what I had expected.

One concern to keep in mind while training a classifier is overfitting. This is the concept where a model gives accurate results for the training data, but not for new data. This is solved by splitting the input dataset in two pieces, a training and a testing set. The training set is used to train the classifier, and the testing set is kept at the side. After training, the test set is used to evaluate the classifier, which gives the final accuracy score.

I ran each classifier through the same set of functions:

```
def run_Kfold(classifier):
    kf = KFold(n_splits=5, shuffle=True)
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,
random_state=42, shuffle=True)
    accs = []
    for train_index, test_index in kf.split(X_train):
        X_train , X_test = X.iloc[train_index,:],X.iloc[test_index,:]
        y_train , y_test = y[train_index] , y[test_index]

        classifier.fit(X_train, y_train)
        y_pred = classifier.predict(X_test)
        acc = classifier.score(X_test, y_test)
        accs.append(acc)
        print('Kfold: ', acc)
    print('Kfold average: ',sum(accs)/len(accs))

    return X_train, X_test, y_train, y_test


def runConfusionMatrix(y_pred):
    cm = pd.DataFrame(confusion_matrix(y_test, y_pred),
                columns=list(df['label'].unique()), index =
(df['label'].unique()))
    sns.heatmap(cm, annot=True, fmt='d');
    print(classification_report(y_test, y_pred))


def findBestHyperParameters(classifier, grid_params, cv):
    gs = GridSearchCV(classifier, grid_params, verbose = 10, cv=cv, n_jobs = -1)
    g_res = gs.fit(X_train, y_train)
    print('Best found score:', g_res.best_score_)
    print('Best found parameters:',g_res.best_params_)
    return g_res.best_params_
```

Figure 26: Common classifier functions

The code fragment in Figure 26 shows the methods used in each classifier's training steps. It starts by running K-Fold cross-validation. This splits the training and testing set into n even parts, and trains and tests the current classifier on each split part. The next method then outputs a cconfusion matrix and a classification report using the classifier's predicted labels (y_pred) against the validation set (y_test). This serves to output a readable result.

The classification report that this second method generates shows precision, recall and f1-scores of the classifier. Precision is the ability of the classifier not to label as positive a sample that is negative, and recall is the ability of the classifier to find all the positive samples. F1-score is a weighted harmonic mean of the precision and recall, where both are equally important.

The confusion matrix serves as a visual of the classification result of the test set. An example output of the K-Fold algorithm and the classification report can be seen below in Figure 27.

```
Kfold:  0.9802371541501976
Kfold:  0.9778656126482214
Kfold:  0.9746835443037974
Kfold:  0.9802215189873418
Kfold:  0.9833860759493671
Kfold average:  0.9792787812077851

Train-test-spilt accuracy :  0.8903416280050611
              precision    recall  f1-score   support

        Bike       0.94      0.75      0.84       546
         Bus       0.96      0.86      0.91       642
       Train       0.82      1.00      0.90       667
     Walking       0.87      0.94      0.90       516

    accuracy                           0.89      2371
   macro avg       0.90      0.89      0.89      2371
weighted avg       0.90      0.89      0.89      2371
```

Figure 27: Output result of classifier training and testing

The second step, hyperparameter tuning, serves to optimize each classifier. This step is unique for each classifier. One example, for the K-Nearest-Neighbor classifier, is to analyze and pick the best value for the n-neighbors parameter, which can be visualized in a graph:

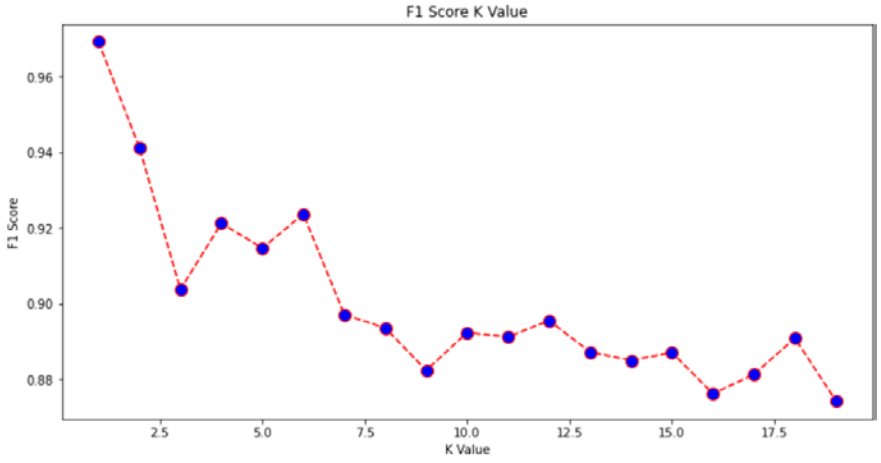

Figure 28: Graph of accuracy values per K value for the K-Nearest Neighbour classifier

For other parameters, GridSearchCV is used, which is a brute force function that finds the best classifier by going over each combination of parameters. This is computationally very expensive.
Once this is done, each classifier is configured with it's best parameters, and used for the manual comparison.

I also run one last sanity check for overfitting, where I evaluate the train and test set against each other using the K-Nearest-Neighbors classifier's n-neighbors. The fact that both sets have roughly the same shape in the graph tells me that the data is not overfit.

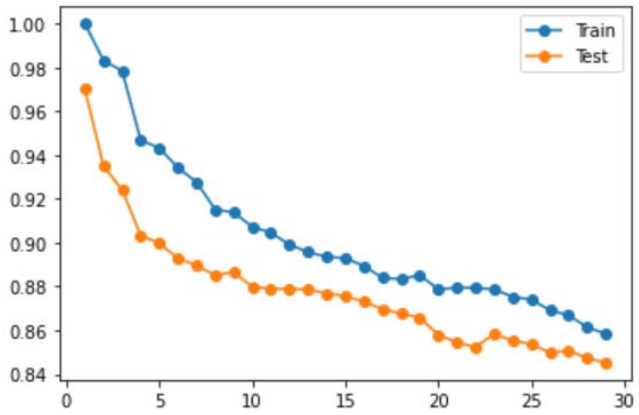

Figure 29: Graph of the accuracy of K-values, comparing the train- and test set

One of the classifiers I made at the end is an ensemble classifier. The idea of this is to use several other classifiers together in order to make a single stronger classifier. I used the Majority Voting Classifier, which functions as it sounds; by having all input classifiers vote on each classification. The quality of this classifier is directly linked to the quality of the input classifiers.

Pipeline

One of my main issues with my workflow is the inefficiency of running all the functions and optimizations manually for each classifier whenever I get new data. In order to solve this, I found the pipeline function. This is a flexible function which allows me to input a set of steps, which are then run automatically. In my case, this means that I can input the whole process of finding the best classifier into my pipeline, saving me significant amounts of time manually testing.

While setting up my pipeline, I also added a couple steps for data preparation. These are scaling data, and variance threshold feature selection. Scaling data, between 0 and 1 for example, can improve the quality of classification, and the variance threshold feature selection serves to eliminate features with low variance (such as features where all values are the same), which in theory also improves the classifier.

The pipeline takes these pre-processing steps into the grid search algorithm, where it adds iterations for using these- or not at all. It runs each iteration 5 times, in order to have consistently accurate results. The code for this pipeline can be found in appendix 2.
The pipeline, with these parameters, goes through 28240 fits, which takes a long time to process (roughly 40 minutes), but does everything I have done manually so far, automatically, making it easy to

run in the background. The result is an output from which I can get the best estimator by directly calling the pipeline object.

```
# Access the best set of parameters
best_params = gscv.best_params_
print('best params: ',best_params)
# Stores the optimum model in best_pipe
best_pipe = gscv.best_estimator_
print('best model: ',best_pipe)

result_df = DataFrame.from_dict(gscv.cv_results_, orient='columns')
print(result_df.columns)
```

✓ 0.1s

```
Training set score: 0.9293203181489516
Test set score: 0.929143821172501
best params:  {'clf__estimator': RandomForestClassifier(bootstrap=False, max_depth=15, min_samples_split=5,
                      n_estimators=20), 'clf__estimator__bootstrap': False, 'clf__estimator__max_depth': 15,
'clf__estimator__min_samples_leaf': 1, 'clf__estimator__min_samples_split': 5, 'clf__estimator__n_estimators': 20,
'scaler': MaxAbsScaler(), 'selector': None}
```

Figure 30: Pipeline result output

I also export the whole output to a csv. From this csv, I can easily find the top n classifiers, which I have then used for the voting classifier. Nevertheless, the Random Forest algorithm provides the best result. This is easily explained by the other algorithms being less precise in the same categories as the Random Forest, resulting in an overall weaker Voting Classifier. Nevertheless, with an accuracy score of around 92-98%, I am happy to continue the implementation using this classifier.

```
Kfold average:  0.9889275529093912

Train-test-spilt accuracy :  0.9232391396035428
              precision    recall  f1-score   support

        Bike       1.00      0.80      0.89       546
         Bus       0.98      0.90      0.94       642
       Train       0.84      1.00      0.91       667
     Walking       0.92      0.99      0.95       516

    accuracy                           0.92      2371
   macro avg       0.93      0.92      0.92      2371
weighted avg       0.93      0.92      0.92      2371
```

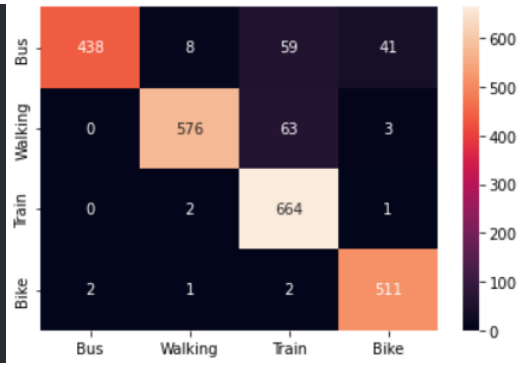

Figure 31: Resulting scores and accuracies of the pipeline

Deep Learning

I also set up a deep learning model. This has been done with TensorFlow by converting my numerical data to a tensor dataset. And setting up a model using several layers.

```
vocab = ['Train', 'Bike', 'Walking', 'Car', 'Bus']
lookup = tf.keras.layers.StringLookup(vocabulary=vocab, output_mode='one_hot')
encoder = LabelEncoder()

numeric_dataset = tf.data.Dataset.from_tensor_slices((np.asarray(X_train).astype('float32'),
encoder.fit_transform(y_train)))
for row in numeric_dataset.take(3):
  print(row)

numeric_batches = numeric_dataset.shuffle(1000).batch(4)

normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(np.asarray(X_train).astype('float32'))

def get_basic_model():
  act_f = 'relu'
  model = keras.Sequential([
    keras.layers.Normalization(axis=1),
    keras.layers.Dense(130, activation=act_f, kernel_initializer='normal'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(100, activation=act_f, kernel_initializer='normal'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(70, activation=act_f, kernel_initializer='normal'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(40, activation=act_f, kernel_initializer='normal'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(20, activation=act_f, kernel_initializer='normal'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation=act_f, kernel_initializer='normal'),
    keras.layers.Dense(1, activation='relu')
  ])

  model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.01),
                loss='categorical_crossentropy',
                metrics=['accuracy'])
  return model

dlm = get_basic_model()
history = dlm.fit(numeric_batches, epochs=10, batch_size=32)
```

Figure 32: Deep learning model setup and fitting

To briefly go over the code used in Figure 32: First, I convert my numerical dataset into tensors so I can run it through Tensorflow. I encode the labels into numbers, and normalize the data.
The actual model has several dense layers made up of a number of nodes to increase complexity and accuracy, and dropout layers in between to avoid overfitting. The rest are fairly standard parameters. However, no matter what I tried to tweak, I could not get the accuracy to reach beyond 30%. The only things I can think of why this happens are the amount of data and the quality of the metrics. For deep learning, a simple rule is that you want at least 10x the datapoints of the number of columns, which in this case would be 130, which I far exceed. The other is metric quality. Dropping my lowest ranking metrics does not impact the final accuracy rating, so I would have to create new metrics if I want that to improve. In the end I remain unsure of the source of this low score. Not knowing exactly how to improve the deep learning set beyond collecting more data or creating the algorithm to use street map data images for training, I will go forward with the regular classifier model.

Prototyping the on-device module: Travelmode Tools

The on-device module, dubbed Travelmode Tools, is the main prototype that this project aims to deliver. It consists of parts of the data collection app, supplemented by an implementation of the classification algorithm, which enables the module itself to classify data without needing to rely on an external service. It will also expose a simplified API, so that apps that implement this library can access the necessary functionality, without needing to do much in terms of configuration.
The machine-learning model is provided encoded in the app release bundle, and new models may be implemented when updating the app. A dynamic delivery system could be considered as a future development step, but is unnecessary for the current prototype.

While rewriting the on-device module to work as a library sounds easy, it has not been without challenges. The first step was to consider what functionality actually needs to be transferred to the Travelmode Tools library. For this, I considered the different parts of the data collection app on their own, and discussing some with the project supervisor, created the following list.

Component	To keep	Reasoning
Data collection	Yes	This is the base functionality that needs to be implemented
Permission checking	Partial	In order to collect data from the GPS, and properly run as a background process, some permissions are still necessary and may need to be requested from the user.
Database connection	No	This was decided along with the project supervisor. The database connection will be left to the implementer.

To go into slightly more detail, permission checking is still done by the Travelmode Tools library. It will attempt to request the necessary permissions from the user, but also allow for the app that implements the module to do so. If the permissions have been granted already, no additional request will be made. If the necessary permissions are not granted, even after the Travelmode Tools library requested it, data collection will not take place.

As for the database connection, while this was a necessary part of the data collector app, the final database implementation is left to the implementer of the Travelmode Tools library. It would be possible to add a database connection, but it would complicate things if the implementer wished to use their own database connection. The possibility of developing both versions- one with a configurable database connection, and one without, was considered, but ultimately deemed impractical. Instead, the Travelmode Tools library has been written to have its own API, with a few simple functions to initialize and stop the data collection.

This solution does mean that the implementer needs to handle the datapoints themselves. In order to do this, I have implemented a data stream. This stream is opened when the Travelmode Tools library is initialized, and can be listened to by the app that implements this library. The Travelmode Tools library sends, when started, a datapoint to this stream every 5 seconds. The implementer can then respond to this event in any way they would like.

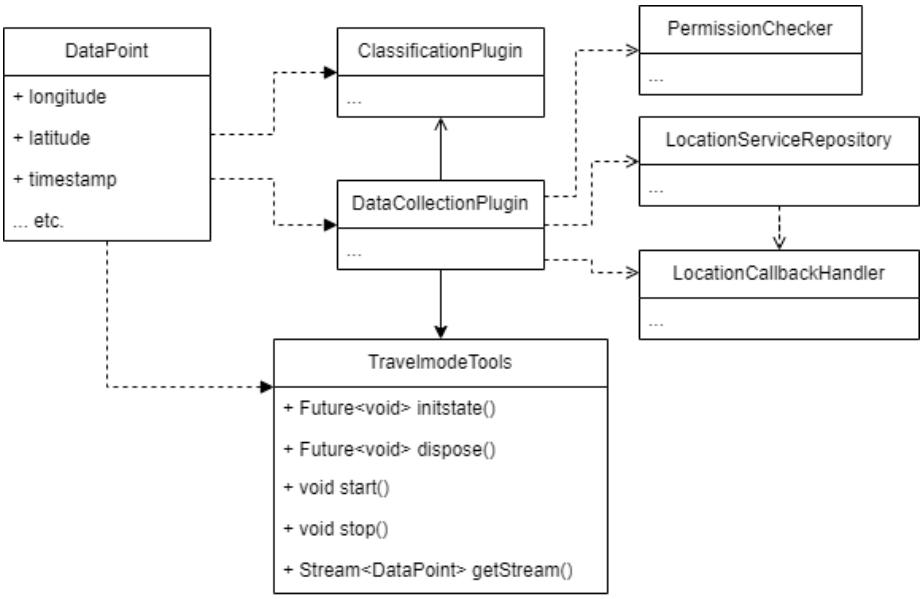

Figure 33: Class diagram of the Travelmode Tools library

I made a small app that implements this library, which shows the ease of implementing this on-device module:

```
DataPoint? point;
late StreamSubscription<DataPoint> dataStream;

@override
void initState() {
  TravelmodeTools().initState();

  dataStream = TravelmodeTools().getStream().listen((DataPoint object) {
    debugPrint(object.toString());
    setState(() {
      point = object;
    });
  });

  getPermissions();

  super.initState();
}
```

Figure 34: Example implementation of my TravelmodeTools library in a Flutter app

The code in Figure 34 is an example implementation of the Travelmode Tools library. When initialized, the app simply needs to call the library's initState function, after which it can listen to the datastream and receive the DataPoint objects.
The main classes in my TravelmodeTools package have their own factory and instance variable, which

means that they do not need to be created as instance variables by the implementer, but are initialized as a unique reference when they are first used by the app. In the code example given in Figure 34, the local 'point' variable will be updated every 5 seconds to show the new point. This DataPoint object received from the datastream also includes the classification result of the machine-learning classifier.

Internally, the library's implementation of the machine-learning classifier is done using its own class. Once exported from the earlier pipeline code, the classifier is imported and used with the SkLite plugin to do classification. The loadClassifier function in Figure 35 only needs to be executed once, but as it is asynchronous, it has been excluded from the factory. After loading the classifier, a simple call to the classify function with the gathered datapoint as an input results in a classified label, which is then converted into an enum for consistency.

```
class ClassificationPlugin {
  late RandomForestClassifier classifier;

  static final ClassificationPlugin _instance = ClassificationPlugin._();
  ClassificationPlugin._();

  factory ClassificationPlugin() {
    return _instance;
  }

  void loadClassifier() {
    loadModel("packages/travelmode_tools/models/2023-05-22-mdl.json").then((x) {
      classifier = RandomForestClassifier.fromMap(json.decode(x));
    });
  }

  ClassificationLabel classify(DataPoint dataPoint) {
    var result = classifier.predict(dataPoint.toClassificationFormat());
    return ClassificationLabel.values[result];
  }
}
```

Figure 35: The ClassificationPlugin Class

One challenge for this implementation has been getting the classifier to work. The SkLite package contained erroneous code, in which a RandomForestClassifier could not be created.
SkLite implements the RandomForestClassifier by creating a number of DecisionTreeClassifiers. These have a different import structure than the RandomForestClassifier, and this is what was causing errors. I had to troubleshoot my way to fixing this package so that it works with the RandomForestClassifier, which it now does.

Building a proof of concept: Route building algorithm

The final piece of what I have built is the route-building algorithm. This was always a bonus prototype, and never a strict goal of the project. I had the time to build a proof of concept, however, and did so.

I used my already-collected data for this proof-of-concept, and split these into routes by looking at the time between nodes of the same device.
Since I collect unique device identifiers, I can separate my dataset by datapoints per user. I can then look at each datapoint and add them to a custom route object, and when the time between the current and next datapoint exceeds a certain threshold around roughly 5 seconds, I cut off the current route and start a new one. This way, I split all datapoints up and add them to their correct route history object. The following code shows the route building algorithm, which uses Route objects that contain a list of Nodes.

```
def build_routes(df):
    prev_timestamp = None
    nodes = []
    routes = []
    for index, row in df.iterrows():
        timestamp = parse_timestring(row['timestamp'])
        node = TravelNode(row['longitude'],row['latitude'],timestamp,row['device_id'])
        if prev_timestamp is None:
            prev_timestamp = timestamp

        if timestamp - prev_timestamp > 60000:
            routes.append(Route(nodes))
            nodes = []

        else:
            nodes.append(node)
        prev_timestamp = timestamp
    return routes
```

Figure 36: Route building algorithm

After collecting these routes, I can look at charting them on a map. For this, I looked at several options.

- Google Maps. The most straightforward solution, but requires an API-key, and thus an account with billing enabled. I wanted to avoid this for a proof-of-concept.
- OpenStreetMap. This is an open-source map with street data in high detail, earlier considered during a research step. I imagined this to be the easiest option, since I could locally download the map of the Netherlands, and chart everything on there. The reality, however, was not as simple.
  As it turns out, OpenStreetMap data consists of several datatypes- not anything presentable as an image.
    - Nodes, which represent a map feature without a size, based on long- and latitude such as a mountain peak.

- Ways, which are a sorted list of nodes, signifying a shape of a linear feature such as roads, rivers, parking areas, jungles, etc.
- Relations, which are sorted lists of nodes and ways, and represents their relation like barriers and u turns
- Tags, which are metadata objects attached to nodes.

As such, I would need a way to interpret this data, and then somehow turn it into a visual representation of the map. Instead of trying to figure this out, I opted for an alternative.

- Folium. One of the first packages I found which I could easily implement. It contains a map of the world, albeit less detailed than OpenStreetMap. On this zoomable and manipulable map, I can add markers, routes, and other highlights of my choice, and I used this to chart the routes. It uses OpenStreetMap in the backend, and such needs an active internet connection, but it saves me from having to write a complex OpenStreetMap implementation myself.

As it is a proof of concept, it is not implemented in a web-UI or anything else, as the final delivery method of this algorithm has not yet been decided by the project manager.

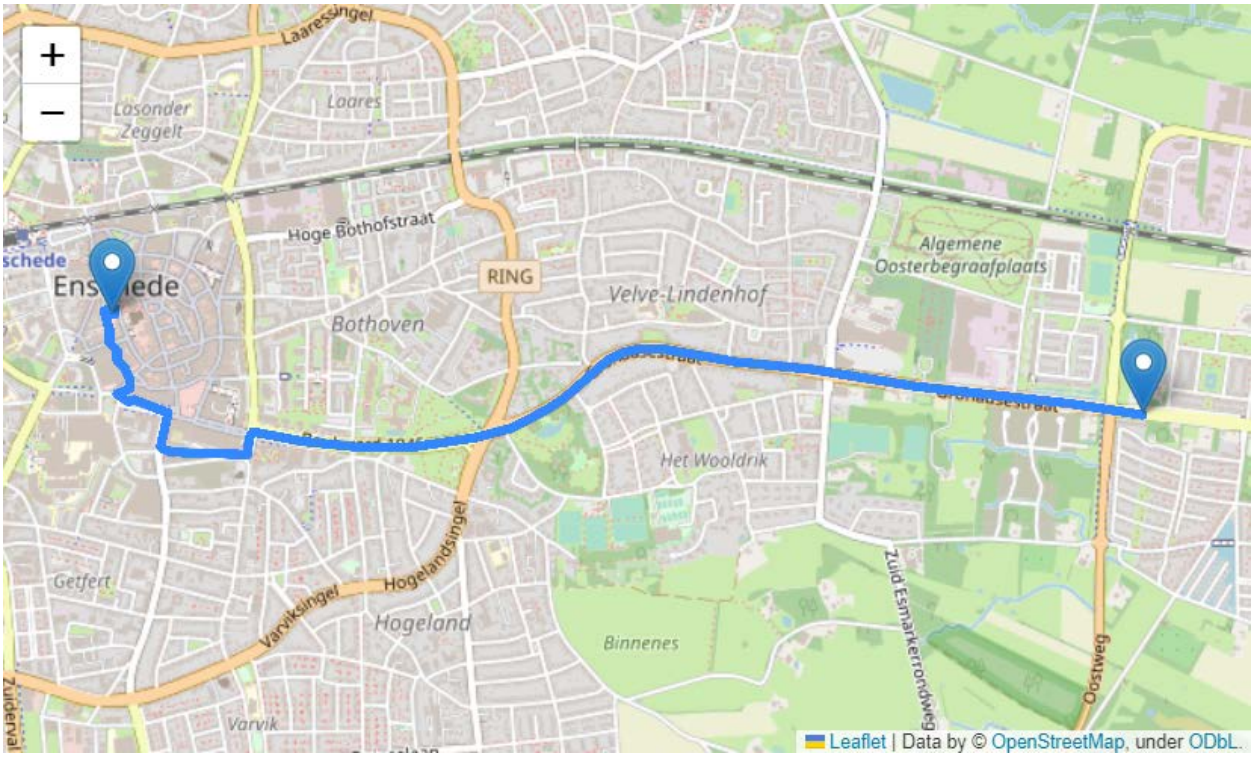

Figure 37: Example output of the route-building algorithm

Result

Research

At the start of the research process, I posed a question:
“How can I predict, with high accuracy, the mode of transportation of someone by using a phone’s sensors, in order to present this in a readable way?”

This question has been answered and the result is presented as a prototype Flutter library.
The direct answer has been to use machine learning, trained on a labelled dataset of points that describe a user’s location. The machine-learning algorithm gives a roughly 93% accurate result between 5 labels. For consistency and ease of use, the library converts this result to an enum.

The route-building algorithm had its own research question:
“How can I correctly segment routes from a set of datapoints, and provide these routes on a visual map?”

This question has been answered by researching several mapping tools, and prototyping these through trial-and error. The final implementation uses a simple Python library. Similar libraries probably exist for a Flutter implementation, but since there was no Flutter requirement for this segment of the project, it is not considered a problem.

Data collection app

The resultant data collection app has been 100% completed. This means that all the requirements as described in Phase 1 have been met. The data collection app functions well, and does what I had set out for it to do. No changes have been made since the development of this app in the earlier phase of the project.

Classification

The requirements for the classification algorithm have been 100% completed. The final implemented model is a Random Forest Classifier. This was picked from the pipeline results as the second-best algorithm. The reason why this is picked over the best algorithm is related to the implementation in the on-device module. The on-device module uses the SkLite library to run classifiers on-device, but the supported models are limited to most base classifiers, and composite classifiers such as Voting Classifiers are not supported. I could not find an alternative library to use, so I had to use the second-best model from my pipeline. This hurts the mean accuracy by 1%, from 93% to 92%, which is regrettable, but acceptable. If the scores were further apart, I would have put effort into finding an alternative.

A couple additional changes were made since the research step on classification. First, I have balanced the data. I suspect that the imbalanced dataset was causing inaccuracies, since some classifiers would fail to classify ‘Walking’, or classify too much as a certain label. It was also causing other inaccuracies which I couldn’t explain; like my training dataset consisting of 60 ‘Walking’ data points, yet the classifier thinking 150 walking points were correctly classified, which was impossible. I am not sure what caused this to happen, but these problems disappeared when I balanced the dataset.

One of the downsides of balancing the dataset was that I was left with relatively fewer samples, since it was limited by the least-represented label. Adjusting the split ratio between the training set and testing set fixed this. The current classifier is trained on about 350 samples per label, totalling in 1759 data points, and tested with about 60 samples per label.
The result is acceptable in my eyes. 92% accuracy is much higher than I was afraid of getting, as I had low expectations, and the accuracy may even be further improved by collecting more data.

Figure 38: Classification result

On-device module

The on-device module was built according to the requirements. 100% of the requirements are met, keeping in mind that the requirements for database connectivity were stripped after consultation.

I have also written an implementation of the created on-device module, to test its functionality. It is a simple application, which just activates and listens to the on-device module, and reports the received datapoints every time a new one comes in. This can be seen in the screenshot below.

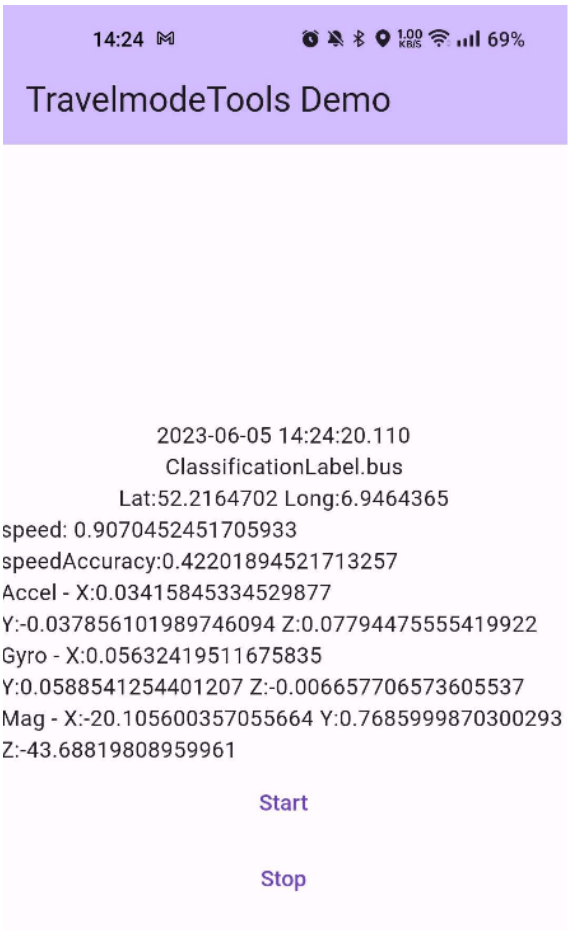

Figure 39: Rough example implemtation of my TravelmodeTools module in Flutter

Route-building algorithm

The route building algorithm was a prototype that was initially planned to be optional, and has been made into a proof-of-concept. The stated requirements have been completed, but a full implementation in such a way that it is usable by a user, either as a web portal or as a mobile application, has not been implemented. This was not part of the project, but could be a follow-up development.
The current implementation is written in Python, but I am confident that methods to do the same in Dart (for Flutter) or JavaScript (for Web) exist.

Heatmap

As discussed before starting the project, the heatmap was an optional prototype that could have been implemented given enough time. However, time ran out, and the heatmap has not been completed. A future proof-of-concept could likely be built using the same mapping functions used in the route-building algorithm. Even less data preparation is likely needed for this, since it simply charts the datapoints themselves. Making the heatmap manipulable (such as filtering by mode of transportation, or time), however, would require some development.

Discussion

After finishing the project, there are some open ends, final concerns, and things I would have done differently, which I discuss here.

Open end: Licencing and Publishing

One open end is publishing the library. I have written it to be implemented locally, but I have not pursued a method to publish it on an online library. This may not be necessary, as it may only be used internally by Baseflow, but it will need a properly written licence regardless. Licencing was something that concerned me during development, but I did not consider it worth the effort to figure out the details around licencing requirements.

Open end: Database

Currently, the data collected by the data-collection app is hosted on a firebase database. This works for free up to a medium volume, but for higher volumes, it may not be optimal. Reading snapshots takes up a lot of the free-to-use daily reads, so this needs to be done carefully. The database implementation of the final library has been left to each implementation, however, so that should not cause issues.

Open end: Route-Building algorithm

This algorithm is currently a proof-of-concept, and lives only as some python code. A future development might see this implemented in a webhosted interface.

Concern: Data volume

The current machine-learning classifier has been trained on an admittedly limited dataset. It shows a proof of concept: it works. In an actual release implementation, however, a method will need to be invented to collect new labelled data. I worry that the accuracy of classification will drop as more users experience edge cases. I am also worried about large scale classification results, even though, in theory, a car traveling along one highway should give similar data to another car on another highway. The fact remains that my data is of a limited area, and I find it hard to predict what may happen on a larger scale.

Change in hindsight: Data-collection app complexity

I have spent a lot of time developing this app, but it will not see much use in the future. I wanted to make it a smooth user experience, and in doing so, made many features that could be considered unnecessary.

One prime example of this is the permissions page and the timer. Sure, the permissions page works great at making sure the user has all the required permissions before allowing them to start the data collection, but the same could have been achieved with a string of popups when the user pressed start, or when the timer detects a revoked permission.

The timer is also superfluous in itself, since it's designed to check for any permissions that are revoked by the user while they run the app, something that only a malicious tester would do, especially considering the small size of the testing team.

I also made functions for viewing and deleting the data generated by a user since I wanted this app to comply with GDPR, but since it is an internal testing app, that was probably unnecessary as well.

In a future, similarly internal prototype, I would strongly consider reigning myself in to only building the necessary functionality.

Appendix

1. Excerpts of 'Handleiding Algemene verordening gegevensbescherming', chapter 3.2

"Persoonsgegevens zijn alle gegevens die:
1) betrekking hebben op;
2) een geïdentificeerde, of;
3) identificeerbare;
4) natuurlijke persoon.

Ad 1) Gegevens die betrekking hebben op
Wil er sprake zijn van persoonsgegevens dan moeten de gegevens allereerst betrekking hebben op een persoon. Met andere woorden: de gegevens moeten over de persoon gaan, ze moeten iets over die persoon zeggen. Wanneer de gegevens niét iets zeggen over een concreet persoon, dan zijn het geen persoonsgegevens."[13] *(p. 24)*

"Ad 2) Geïdentificeerde
Gegevens hebben alleen betrekking op een natuurlijke persoon wanneer deze geïdentificeerd is of identificeerbaar is. Een persoon is geïdentificeerd wanneer deze uniek van alle andere personen binnen een groep te onderscheiden is. Een persoon is identificeerbaar wanneer deze nog niet geïdentificeerd is, maar dit zonder onevenredige inspanning wel mogelijk is."[13] *(p. 24)*

"Personen kunnen ook geïdentificeerd worden op basis van andere, minder directe identificatoren. Denk hierbij aan uiterlijke kenmerken (lengte, postuur en haarkleur), sociale en economische kenmerken (beroep, inkomen of opleiding) en online identificatoren zoals IP-adressen. Hoewel deze gegevens op zichzelf ons meestal nog niet in staat stellen om een persoon te identificeren, kunnen zij door hun onderlinge samenhang of door koppeling aan andere gegevens alsnog leiden tot identificatie. We spreken daarom van indirect identificerende gegevens.

Of iets een persoonsgegeven is voor u, is dus afhankelijk van de vraag of het gegeven of de gegevens die u verwerkt u in staat stellen om iemand direct of indirect te identificeren (uniek te onderscheiden binnen een groep). Wanneer de persoon nog niet geïdentificeerd is (wat doorgaans het geval is als u geen direct identificerende gegevens verwerkt) moet u bepalen of de persoon niet alsnog identificeerbaar is."[13] *(p. 25)*

"Ad 3) identificeerbaar
Een persoon is identificeerbaar indien zijn identiteit nog niet is vastgesteld, maar dit redelijkerwijs, zonder onevenredige inspanning, wel kan gebeuren. Dit gebeurt meestal op de volgende wijze:
• gegevens worden gekoppeld aan direct identificerende gegevens; of
• gegevens zijn door hun onderlinge combinatie dusdanig uniek dat ze maar op één persoon betrekking kunnen hebben."[13] *(p. 25)*

[13] *Schermer, B. W., Hagenauw, D., & Falot, N., Handleiding Algemene verordening gegevensbescherming en Uitvoeringswet Algemene verordening gegevensbescherming (2018). Ministerie van Justitie en Veiligheid.*

"Bij de beoordeling of er sprake is van identificeerbaarheid moeten de mogelijkheden van de verwerkingsverantwoordelijke (of een derde) om de identificatie tot stand te brengen worden meegewogen. Het gaat dus niet om de hypothetische mogelijkheid dat gegevens gekoppeld of gecombineerd kunnen worden, maar om de vraag of de verwerkingsverantwoordelijke dit zonder onevenredige inspanning kan."[13] *(p. 25)*

"Ad 4) natuurlijke persoon
De Verordening is alleen van toepassing op de verwerking van gegevens over natuurlijke personen. Gegevens over organisaties (ondernemingen en dergelijke) zijn géén persoonsgegevens,"[13] *(p. 25)*

2. Pipeline code

```
pipeline = Pipeline([
    ('scaler', None),
    ('selector', None),
    ('clf', ClfSwitcher()),
])

parameters = [
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(), Max-
AbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator':[tree.DecisionTreeClassifier()],
        'clf__estimator__min_samples_leaf': [1, 2, 3, 4, 5],
        'clf__estimator__criterion': ['gini', 'entropy'],
        'clf__estimator__max_depth':
[2,4,6,8,10,12,14,16,17,18,19,20,22,24,26,28,30,32,50,100],
    },
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(), Max-
AbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator': [KNeighborsClassifier()],
        'clf__estimator__n_neighbors': range(1,20),
        'clf__estimator__weights': ['uniform','distance'],
        'clf__estimator__metric': ['minkowski','euclidean','manhattan',
'cosine'],
    },
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(), Max-
AbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator': [GaussianNB()],
        'clf__estimator__var_smoothing': np.logspace(0,-9, num=100),
    },
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(), Max-
AbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator': [svm.SVC()],
```

```
        'clf__estimator__C': [0.1, 1, 10, 100,1000],
        'clf__estimator__gamma': [1, 0.1, 0.01, 0.001, 0.0001],
        'clf__estimator__kernel': ['rbf'],
    },
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(),
MaxAbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator':[RandomForestClassifier()],
        'clf__estimator__bootstrap': [True, False],
        'clf__estimator__max_depth': [15,16,17],
        'clf__estimator__min_samples_leaf': [1, 2, 4],
        'clf__estimator__min_samples_split': [2, 5, 10],
        'clf__estimator__n_estimators': [19,20,21],
    },
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(),
MaxAbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator':[QuadraticDiscriminantAnalysis()],
        'clf__estimator__reg_param': [0.00001, 0.0001, 0.001,0.01, 0.1],
        'clf__estimator__store_covariance': [True, False],
        'clf__estimator__tol': [0.0001, 0.001,0.01, 0.1],
    },
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(),
MaxAbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator':[SGDClassifier()],
        'clf__estimator__alpha':[0.00001, 0.000001],
        'clf__estimator__penalty':["l2", "elasticnet"],
        'clf__estimator__max_iter':[10,50,80,100,500,1000],
    },
    {
        'scaler': [None, StandardScaler(), MinMaxScaler(),
MaxAbsScaler()],
        'selector': [None, VarianceThreshold()],
        'clf__estimator':[VotingClassifier(voting='soft', estimators=[
                ('Random Forest',RandomForestClassifier(bootstrap=
False, max_depth= 17, max_features= 'sqrt',
                    min_samples_leaf= 1, min_samples_split= 5,
n_estimators= 21)),
                ('KNeighbors',KNeighborsClassifier(n_neighbors= 1,
weights = 'uniform', metric = 'manhattan'))]),
            VotingClassifier(voting='soft', estimators=[
```

```
                ('Random Forest',RandomForestClassifier(bootstrap=
False, max_depth= 17, max_features= 'sqrt',
                    min_samples_leaf= 1, min_samples_split= 5,
n_estimators= 21)),
                ('KNeighbors',KNeighborsClassifier(n_neighbors= 1,
weights = 'uniform', metric = 'manhattan')),
                ('Decision
Tree',tree.DecisionTreeClassifier(min_samples_leaf=1,criterion='entropy'
,max_depth=32))]),
            VotingClassifier(voting='soft', estimators=[
                ('Random Forest',RandomForestClassifier(bootstrap=
False, max_depth= 17, max_features= 'sqrt',
                    min_samples_leaf= 1, min_samples_split= 5,
n_estimators= 21)),
                ('KNeighbors',KNeighborsClassifier(n_neighbors= 1,
weights = 'uniform', metric = 'manhattan')),
                ('Decision
Tree',tree.DecisionTreeClassifier(min_samples_leaf=1,criterion='entropy'
,max_depth=32)),
                ('Quadratic Discriminant',
QuadraticDiscriminantAnalysis(reg_param= 0, store_covariance= True, tol=
0)),
                ('GaussianNB',GaussianNB(var_smoothing=
0.005336699231206))]),],
    }
]

gscv = GridSearchCV(pipeline, parameters, cv=5, n_jobs=16,
return_train_score=True, verbose=10)
gscv.fit(X_train, y_train)
```