This chapter will provide an introduction to the thesis by laying out the motivation and background as well as the problem statement and the goals which were set out to be achieved. In addition the results and outcome will be briefly discussed and lastly an overview of the structure of the thesis will be presented.

Most people walk around with a very powerful computer in their pocket, that contains a variety of sensors capable of painting a picture of the user's current context. A user's context is a set of digital characteristics, also called features, which arise from an individual interacting with their environment, such as when moving around between places in one's everyday life. This thesis will describe the design and implementation of a software package for the Flutter framework capable of generating a fully-automated mobility context in real-time, by tracking location data on a smartphone. The package will be part of the Carp Mobile Sensing Framework (CAMS) which provides an array of digital phenotyping capabilities to an application developer.

The research problem addressed in this project is focused on the software engineering aspect of developing a digital phenotyping library and relates to the *re-usability of source code* as well as the current *lack of support for real-time computation*.

Firstly, there have been numerous contributions within the field of using smartphone data for phenotyping and predicting the user's state for many years. However, much of the research conducted, specifically within the field of user-mobility context generation, has been done without publishing source code in a way such that it may be used by other researchers in the future. Mainly this comes down to researchers not publishing the source code at all, due to source code not being the focus of their research. Sometimes the source code is released, however, but then over time suffers from an (understandable) lack of maintenance and thus becomes obsolete. Furthermore, researchers tend to focus on only the Android platform since this is the more

prevalent platform of choice in many countries, however, this is not a good research practice for reasons we will elaborate on later.

In addition, the existing contributions gather all of the data before feature computation is carried out. This has the clear advantage of making it easier to run and evaluate the mobility feature algorithms. However, this also has the disadvantage that the feature generation is not possible in real-time, where they are the most useful. Certain features rely on prior contexts being computed and readily available and are therefore easily implemented when done in an off-line fashion on a desktop computer. However, in a real-time situation on a phone, these prior contexts will need to be stored on a device that has limited storage, and the algorithms must be adjusted to read in these stored set contexts. The proposed software package will solve the problem of implementing these algorithms from scratch as well as allow the application programmer to compute the user's mobility context in real-time.

Researchers within the field of mental health illnesses from a health-tech background will benefit greatly from using a Flutter package with a high level of abstraction, in which the researcher's focus is moved from software engineering to data analysis and study design. Currently, researchers often choose to start from scratch which is a laborious task that takes up valuable time which could be spent on more relevant work. Furthermore, since Flutter is capable of compiling to both iOS and Android source code, studies need not be restricted to a single mobile platform, hence why Flutter was the app-framework of choice.

An additional benefit of computing features on the phone itself, in contrast to computing it on a central server or desktop, is the anonymization of GPS data; the generated context contains information that is much less sensitive and therefore has implications on privacy as well as GDPR compliance.

Major Depressive Disoder (MDD), commonly known as depression, is a serious illness that is resource-intensive to treat using traditional methods such as face to face consultations. This costs society a lot of money, and it is, therefore, worthwhile to explore treatments that can improve the quality of life of these patients.

Among many treatments, Behavioural Activation is very well supported by mobile sensing technologies which have become a growing field due to the rise of ubiquitous computing.
Currently, the diagnosis in bipolar disorder, also known as manic depression, relies on manual patient information and clinical evaluations and judgments with a lack of objective testing. Some of the essential clinical features of individuals suffering from bipolar disorder are changes to their daily behavior [**objective_smartphone_data_as_diagnos**
These changes in behaviour can be captured by the user's *mobility context*, and can thus provide very helpful in customizing a Behavioural Activation treatment. A mobility context can be generated by user rules (i.e., manual), and triggered by events (i.e., system, user activity) from semi- or full- automated algorithms. Here, it will be investigated how the context can be generated in a fully-automatic, real-time fashion.

Clustering location data into places has been done previously in a more general context, for example to learn significant locations and predicting where the user will move next [**learning_significant_locations**] or to learn whether or not low accuracy sampling can be used to identify these locations [**sparse-location-2014**]. Recently, location data has also been used within the context of depression, by calculating certain features such as the percentage of time spent at home [**Saeb2015**; **Canzian2015**]. In addition [**Saeb2015**] also showed that certain mobility features correlate strongly with scoring highly on the Patient Health Questionnaire (PHQ-9) (see Appendix **??**), which indicates an individual is depressed.
In regards to Behavioural Activation, D. Rohani developed two recommender systems for treating depression while working on his PhD at CACHET [**mubs-rohani**; **moribus**], both of which rely on manual user input, and it is a future improvement to incorporate location data into the recommendation-algorithm. Lastly, the work done at CACHET on the CARP Mobile Sensing Framework[1] within mobile sensing provides an ecosystem in which the resulting package would fit in.

Existing contributions within the field of generating mobility features and context are cumbersome, if even possible, to reproduce with regards to their source code. Furthermore, the algorithms that exist in the literature do not necessarily lend themselves to the real-time computation of features. How these two things can be achieved will be investigated in this thesis and are addressed in the following research questions:

[1] `https://github.com/cph-cachet/carp.sensing-flutter/wiki`

Question 1: *Which mobility features are relevant to include in a software package?*

Question 2: *How can these features be computed in real-time, on a smartphone device?*

Question 3: *How does the API design of such a software package look like?*

The research goals closely match the research questions and will explain how the different sub-questions are answered, and which methods are used to do so.

The paper [**Saeb2015**] describes a list of mobility features, some of which they prove to strongly correlate with depressive behavior. These features need to be implemented in their most simple form, i.e. off-line, where all the location data for a given day is readily available. This will be carried out in Python with a synthetic dataset and later in Dart. This is done in order to demonstrate the quality of the algorithms. A pre-processing procedure described in [**sparse-location-2014**] producing intermediate features has already been carried out by Jonas Busk prior to starting this thesis. From these intermediate features, many of the mobility features can be easily derived.

The features described above will be implemented such that they can be evaluated on a partial dataset, i.e. an incomplete day's data. Many of the features will have minor changes applied to their calculation, however the *Routine Index* feature as defined by [**extraction-of-behavioural-features**] and [**Saeb2015**] (here called the *Circadian Movement*), may prove to be a challenge to implement in real-time. By its basic definition, this feature lends itself to be computed only once several full days of data have been collected. This makes the feature non-trivial to implement in a real-time fashion where it may computed at any time.

An algorithm/procedure that runs the Mobility Feature algorithms on a smart-phone has to be written which involves a couple of steps. Firstly, location data has to be collected and stored on the smartphone, and then it has to be demonstrated that features can be computed whenever they need to be. However, storing raw data every day would end up taking up too much disk space and is therefore not an ideal solution when working with a smartphone. This means the solution has to be able

to compute these history-dependent features without relying on storing raw data of each day.

The software package should be designed according to the Flutter packages best practices [2], be properly documented and be released on the Dart package manager [3].

To demonstrate how the software package containing the algorithm will be used in practice, a Flutter demo-app will be developed in parallel with the software package. The functionality of the application should be in accordance with the contents of *Goal 3*. By developing an actual application and seeing the algorithms in use, the structure of the API should emerge such that it is 'nice to use' from an application developer's point of view.

To test the quality of the Mobility Features in action, a small scale study of 5-10 participants should be conducted and should run for 2-4 weeks (the longer the better). The participants will use the demo application. Also, in order to validate the quality of the features produced daily during these weeks, the participants will have to fill out a small diary in order to compare the results of the algorithms with the subjective experience of the users. This diary will be a part of the demo application. Preferably the users should be reminded daily to fill out the diary such that as much user data is collected as possible.

In this section, it will be addressed to which extent the research questions and goals were met.

The features described in [**Saeb2015**] and [**extraction-of-behavioural-features**] were implemented in Dart by using standard imperative language features such as loops and if-else clauses. Specifically, a set of intermediate features, namely *Stops*, *Places*, and *Moves*, were also implemented which aided in reducing the sheer amount of data processed by the feature-extraction algorithms.

[2] `flutter.dev/docs/development/packages-and-plugins/developing-packages`
[3] `www.pub.dev`

The most important features pertaining to depression were *Home Stay*, *Entropy/Normalized Entropy*, and the *Routine Index*. All features except for the *Routine Index* can be evaluated on a daily basis without the need for historical data, and as such those features were implemented first and the *Routine Index* was implemented later. In addition to the features described by [**Saeb2015**], most of which are evaluated daily, the features were also extended to aggregate features which are a daily feature averaged over a period of the last 28 days. The *Routine Index* features was chosen to be put in Among these aggregated features.

The Routine Index as well as the aggregate features, are calculated by storing the previously mentioned intermediate feature *Stops* on the device. These features effectively serve as a compressed form of the raw dataset, which is saved on the device. The compressed dataset takes up three orders of magnitude less disk space compared to the corresponding raw data, which also made it possible to calculate those features which relied on historical data.

All the implementation details of the algorithms were hidden away from the programmer and a high abstraction level API was provided, with thorough documentation on its usage. Even the problem of saving intermediate features on the device, such that they may be used in the future, was handled with minimal need for the programmer to be involved. In the process of developing this app, the quality of the *Mobility Features Package* was also improved both in the accuracy of the features but also in terms of the abstraction level of the API. However, a few abstraction-level decisions are still up for debate with regards to how *open* the API should be, and whether or not the intermediate features should be part of it since they themselves contain sensitive information.

A field study with 10 participants was conducted in which a study app was developed in Flutter, in order to test the usage of the package. Due to Google restricting the Location API on Android shortly after the thesis plan was laid out, the study was conducted using only iPhones. This was done in order to not spend too much time on things out of our control. During the development of the application, the COVID-19 pandemic hit Europe, and therefore it was not possible to meet up with most of the participants physically. The demo application, therefore, had to be distributed digitally which was done via Apple's Testflight service, which is a good idea in any case, since it allows for quick patching of bug-fixes with minimal installation trouble to the users.

This section will provide an overview of the thesis in the form of a short summary of each remaining chapter.

In this chapter, the most relevant research with regards to ubiquitous computing and generating mobility features will be laid out. This includes relevant contributions within the broader context of mobile sensor context generation, algorithms for pre-processing spatial data, clustering GPS data, how features can be generated from geospatial clusters, and lastly what the features can be used for.

Here, it will be discussed which requirements the final product should fulfill. This includes the mathematical definitions and the data model for the *Mobility Features*. This includes the algorithms for computing these, and how some of these algorithms have been adapted such that they work on an incomplete, real-time generated dataset.

Here, it will be discussed how the algorithms were concretely implemented both in Flutter such that they fulfill the functional requirements outlined in the *Algorithm Design* chapter.

Here, it will discus how the implemented algorithms were wrapped up into a software package with a public-facing API. This includes public and private data, API considerations and documentation. Furthermore an integration to CAMS will also be discussed.

Validation of the package will be discussed both from a quantitative- as well as a qualitative perspective. The quantitative validation will verify that the algorithms produce meaningful and correct results, which will be validated by means of unit testing as well as data analysis in which subjective data reported by participants will be compared to the features computed by the algorithms. The qualitative part will be much more focused on how the software package was designed, i.e. the ease of use for the application developer as well as design of the API. Concretely this can be

evaluated in terms of the number of lines an application developer needs to write in order to use the package.

Here the results will be discussed as well as how one may go about developing the software package further, including changing the existing algorithms and which design- and implementation choices could have been made differently. This includes how the API is designed with regards to trade-offs between varying the abstraction-level, complexity and openness of the API.

The most important findings and things that could be worked on in the future will be described in this chapter.

This chapter will describe what work has been done within the treatment of depression related to digital phenotyping and context generation. Secondly, a series of previous contributions will be presented which deal with computing mobility features including how these features can be used in a medical context. Thirdly, a brief insight into existing mobile sensing frameworks will be given, including how the *Mobility Features Package* will fit into one of these frameworks. Lastly, an example of a recommender system is given for which the *Mobility Features Package* is an ideal use case.

The paper [**digital_phenotyping**] deals with the topic of collecting and aggregating user data into a so-called digital phenotype. It is predicted that by 2050, the biggest impact in psychiatry and mental health will have been the revolution in technology- and information science. Smartphones have become ubiquitous in the past decade and there are over three billion smartphones with a data plan worldwide, each of which has computing power which surpasses supercomputers of the 1990s. In areas around the world without easy access to clean water, ownership of a smartphone and by proxy, rapid access to information has become a symbol of modernity.

In the realm of psychiatry, a current data collection problem is the dependence on self-reporting of sleep, appetite, and emotional state, even though it is recognized that depression will impair people's ability to remain objective in assessing their own behavior and thus data is prone to be faulty [**digital_phenotyping**]. Another current problem is how it how people suffering from mental illness tend to not seek help before it is too late. Depressive relapses are therefore also often reported with considerable delay for patients currently in treatment.

The smartphone offers an objective form of mental-state measurement which is referred to was *Digital Phenotyping*, which uses built-in sensors such as geo-location, accelerometer, and human-computer interactions (HCI) to infer the state of the patient. This makes it possible to assess people by using data in a real-time fashion, rather than in retrospect as is currently is done. Digital phenotyping could in theory fill the role of a smoke detector which provides early signs of relapse and recovery, without replacing the face-to-face consultations entirely. In addition, this also al-

lows researchers to track patients in their own environment, rather than in a clinical environment.

However, when does measurement become surveillance? Is phenotyping by using data such as geo-location too invasive? A series of ethical issues have to be addressed before digital phenotyping can realistically hope to be employed as a tool for population health. Although, some of these issues have technical solutions, such as with tracking human-computer-interactions which is mostly related to *how* the user interacts with a device, i.e. the pacing with which is typed, scrolled or how long phone calls last - rather than the content of what is provided to the device i.e. what is typed, spoken during a phone call, or the websites visited.

This thesis by [**learning_significant_locations**] from 2002 is one of the earliest contributions to present a system that uses GPS data to infer user context and use this context.

For data collection, a wearable GPS receiver was used which tracked user location for four months with an accuracy of 15 meters, meaning the same physical location may have resulted in slightly different GPS coordinates from day today. For pre-processing data some of the data was discarded by setting the sensor to only log data while the user was traveling at a speed greater than 1 mile per hour, with the average human having a walking speed of about 3 miles per hour.

Given a data set of GPS points, a *significant place* is found by examining data and finding periods of low user-movement. The duration of the period was (somewhat arbitrarily) chosen to be 10 minutes. Due to the 15 meter accuracy of the sensor, the location coordinates found may be somewhat noisy from day today. In order to identify the mean of these noisy place locations, a modified version of the *K-means* clustering algorithm was applied to the dataset, with a radius parameter. The optimal radius for the clustering algorithm is found by examining the graph of the number of clusters found (K) as a function of the radius. The knee of the graph (see figure **??**) signifies the radius just before the number of locations begins to converge to the number of clusters found. This procedure effectively figures out which significant places belong to the same *location* and which are outliers points which should be

considered noise, i.e. they either do not belong to a location or there is too low confidence to say whether they do or not.

Within a location, several *sub-locations* may exist, which are perhaps best exemplified by a university campus, as a location, having several buildings for different departments, i.e. *sub-locations*. To find these, all *significant place* data-points belonging to a *location* is given as input to previously described clustering algorithm. The optimal radius is once again found by looking at the knee on the graph and from this optimal parameter choice, the *sub-locations* within a given location are identified.

A Markov model was set up with each state representing a Location and transitions representing real-life transitions between locations. To compute the probability of a path, or a transition between places, the relative frequency of the path was computed.

The DBSCAN algorithm is an essential algorithm for clustering GPS data points and was published by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu in 1996 [**density-based-1996**]. The core concept of DBSCAN is to cluster location data based on a density measure rather than a pure distance measure, such as is

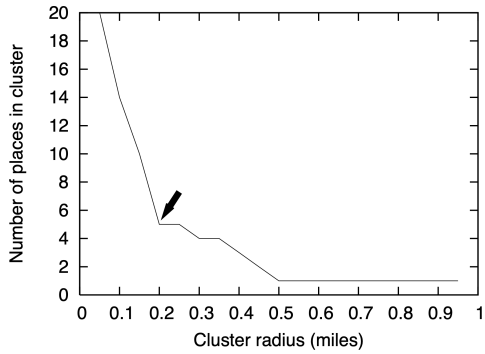

Figure 2.1: The number of places found in the dataset the location radius parameter changes. The arrow indicates the knee in the graph at which point the radius will be used to find *sub-locations*. Source [**learning_significant_locations**].

the case with the *K-means* clustering algorithm. This allows the clusters to take on nearly any shapes and sizes, in contrast to round shapes which are found by using *K-means* and *Gaussian Mixture Model*, see figure **??** for examples.

GPS data has a higher density inside clusters than outside the clusters and the density in noisy areas is lower than density in clusters. Here, *noisy* refers to points that are spread randomly within some areas and do not cluster around a centroid.

The DBSCAN algorithm will, given a set of geospatial data points and a small set of parameters, find these dense clusters, as well as noisy data points, where the clusters correspond to places and the noisy data points, are outliers which do not belong to a place. The output of the algorithm is labeling of each point in the input dataset as either belonging to a cluster or being a noisy data point.

The DBSCAN paper defines a *Neighbourhood* as a collection of data points within some radius, i.e. an area defined by a distance function. Within a *Neighbourhood*, two types of points can reside: *Core points*, that is, points which are inside in the neighborhood, and *Border points* which lie on the edge of the neighborhood and delimit it.

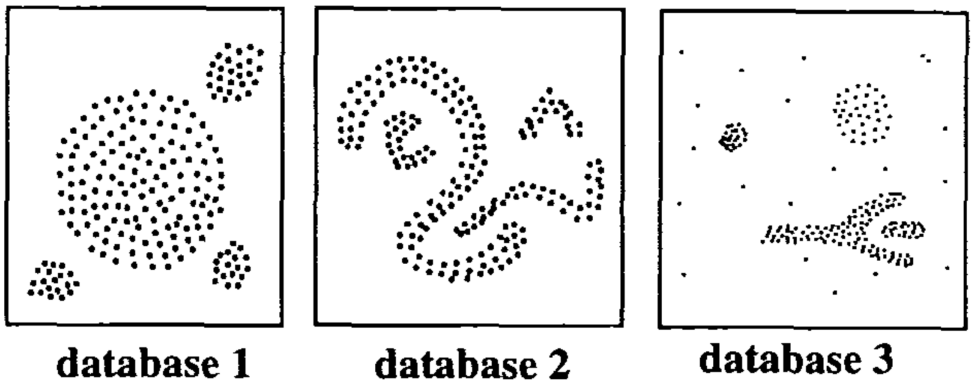

Figure 2.2: Data from three different databases with very distinct cluster shapes. Database #1 has very round clusters which would identified correctly with K-means whereas database #2 and #3 have clusters which would require a different clustering approach. Source: [**density-based-1996**].

An *Epsilon Neighbourhood* is a *Neighborhood* defined on a point p which includes all the points q which are inside a radius of ϵ of p. Formally this is defined as:

$$N_\epsilon(p) = q \in D \quad | \quad dist(p,q) \leq \epsilon$$

Note: The N_ϵ of a border point contains much fewer points than that of a core point.

We Require for all points p and a given cluster C: There must be a a point $q \in C$ st. $p \in N_\epsilon(q)$ of and $N_\epsilon(q)$ contains at least T_{min} points where T_{min} is a parameter which defines the minimum number of points required.

A point p is *Directly Density-Reachable* from point q wrt. the parameters ϵ and T_{min} if $p \in N_\epsilon(q)$ and $|N_\epsilon(q)| \geq T_{min}$ (*Core Point Condition*).

This implies a border point p may be *Directly Density-Reachable* from a core point q, but it is likely, not true the other way around, since $N_\epsilon(p)$ is probably smaller than the minimum amount of points needed to satisfy the *Core Point Condition*.

A point p is *Density-Reachable* (DR) from a point q if there exists a chain of points $p_1, ..., p_n$, where $p_1 = q$ and $p_n = p$, such that p_{i+1} is *Directly Density-Reachable* from p_i. This relation is denoted $DR(p,q)$.

Note that, only symmetric for core points, but not symmetric in general and that two border points from the same cluster may not be Density-Reachable due to them not satisfying the Core Point Condition.

A point p is *Density-Connected* (DC) to q if there exists a point o st. both p and q are *Density-Reachable* from o, this is denoted $DC(p,q)$.

Density-Connected is a symmetric relation meaning that $DC(p,q) \iff DC(q,p)$.

A cluster is a set of *Density-Connected* points which is have maximal *Density-Reachability*.

That is, for all points p, q, if $p \in C$ and $DR(p, q)$ then $q \in C$.

For all points $p, q \in C$ it holds that p is *Density-Connected* to q, i.e. $DC(p, q)$.

The parameters ϵ and T_{min} are global, i.e. the same for all clusters.

Noisy points are defined as all the points p for which $p \in D \wedge p \notin C_i$ for all clusters $i = 1, ..., N$, that is, all the points not belonging to any cluster.

Let p be a point in D for which the core point condition holds. Then, the set O which is the set of *Density-Reachable* points from p (wrt. the parameters) is a cluster wrt. to the parameters. This means, a cluster C contains exactly the points which are *Density-Reachable* from an arbitrary core point of C.

Let C be a cluster wrt. to the chosen parameters and let p be a core point in C, then it holds that $C = O$.

There is no reliable way of knowing the parameters ϵ and T_{min} for each cluster in advance. Instead, the parameters are determined based on the least dense cluster and used globally for all clusters.

Start with a random point p and get all *Density-Reachable* points from p wrt. the parameters. Given $p \in D$:

- If p is a core point, then $O = DR(p)$ is a cluster (*Lemma 2*).
- Otherwise, if p is a border point, then $DR(p) = \{\}$ and the algorithm proceeds to the next point q

Once all points have been considered, the algorithm terminates.
Note: Since the parameters are global, the algorithm may merge two clusters according to *Definition 5* if two clusters are close to each other (even if the two densities are different).

This paper by Saeb et al. published in 2015 [**Saeb2015**] forms the basis for the context generation of this thesis, with regards how one may derive mobility features from GPS data with the features *Location Entropy* and *Circadian Movement* being new contributions at the time at which the paper was written. In addition, this paper also shows how certain features, in particular, correlate strongly with a *PHQ-9* questionnaire score, which is relevant in a mental health context. Since the *PHQ-9* questionnaire requires manual input from the user, and the user may forget to fill out the questionnaire, the strong correlation is great news. This could imply that it is possible to automate the patient data gathering process by simply tracking the location of the patient with a high frequency, and thus get around the issue of manually gathering biweekly subjective questionnaire data. The *PHQ-9* questionnaire (see Appendix **??**).

The pre-processing of GPS data is split into two phases: Firstly, each data point is labeled as either being in a stationary or transitional state, for which the time-stamp of the current, the previous, and the next data point is used. By using this time difference as well as the distance between points the average velocity is calculated. The labeling is afterward done by using a threshold of 1 km/h; a speed lower than the threshold indicates the state is stationary and higher is a transitional point. For analyzing the data further, only the stationary points are considered. Secondly, the stationary points are processed using k-means clustering to identify frequently visited places. Since the number of places, i.e. the number of clusters for the *K-means* algorithm (the parameter K) is not known beforehand, the best parameter value is found using cross-validation. Concretely the algorithm for finding K is to increase K until the largest cluster found has a radius of 500 meters or lower.

After the pre-processing step, the following features can be computed:

Number of Clusters: This feature represents the total number of clusters found by the clustering algorithm.
'Cluster' and 'Place' may be used interchangeably when describing features. A cluster is simply the mathematical term for a collection of data points which corresponds to a place with some GPS coordinates in the real world.

$$N = K$$

Location Variance (LV): This feature measures the variability of a participant's location data from stationary states. LV was computed as the natural logarithm of the sum of the statistical variances of the latitude and the longitude components of the location data.

$$LV = \log(\sigma_{lat}^2 + \sigma_{lon}^2 + 1)$$

Location Entropy (LE): A measure of points of interest. High entropy indicates that the participant spent time more uniformly across different location clusters, while lower entropy indicates the participant spent most of the time at some specific clusters. Calculated as

$$Entropy = -\sum_{i=1}^{N} p_i \cdot \log p_i$$

where each i represents a location cluster, N denotes the total number of location clusters, and p_i is the percentage of time the participant spent at the location cluster i.

Normalized LE: Normalized entropy is calculated by dividing the cluster entropy by its maximum value, which is the logarithm of the total number of clusters.

$$NormalizedEntropy = \frac{Entropy}{\log N}$$

Normalized entropy is invariant to the number of clusters and thus solely depends on their visiting distribution. The value of normalized entropy ranges from 0 to 1, where 0 indicates the participant has spent their time at only one location, and 1 indicates that the participant has spent an equal amount of time to visit each location cluster.

Home Stay: The percentage of time the participant has been at the cluster that represents home. We define the home cluster as the cluster, which is mostly visited during the period between 12 am and 6 am.

Transition Time: Transition Time measures the percentage of time the participant has been in the transition state.

Total Distance: This feature measures the total distance the participant has traveled in the transition state.

Circadian Movement: This feature measures to what extent the changes in a participant's location follow a 24-hour, or circadian, rhythm. To calculate the circadian movement, we obtained the distribution of the periodicity of the stationary location data and then calculated the percentage of it that falls in the 24±0.5 hour periodicity. However, the paper does not describe in great detail how this feature is implemented.

The features which correlated strongest with the *PHQ-9* scores over two weeks were the following:

Circadian Movement: This features correlated negatively with the *PHQ-9* score, and can be interpreted as depressed individuals tend to have less of a routine than healthy individuals.

Location Variance & Normalized Location Entropy: These features have a similar meaning and both correlate negatively with the *PHQ-9* score. The takeaway here is likely that visiting very different places every single day can be seen as a sign that the individual is depressed, which also correlates with the Circadian Movement, since visiting different places every day results in a low level of routine.

Home Stay: This feature had a Strong positive correlation with the *PHQ-9* score, and thus the main take away from this feature is that spending a lot of time at home is an indicator of being depressed.

The paper finds that depressive symptoms tend to change slowly over weeks, with little day-to-day variation and thus it makes more sensors to group data on a weekly basis. This may explain why two weeks of sensor feature correlated more strongly with the *PHQ-9* than either daily sensor features or EMA measures. Unlike EMA ratings, which are momentary, the *PHQ-9* assessment reports symptoms over a period of two weeks. The study conducted suggests that it is possible to monitor depression passively using phone sensor data, and in particular, GPS. Most people are unwilling to answer questions repeatedly over long periods of time, while passive monitoring could improve the management of depression in populations, allowing at-risk patients to be treated more quickly as symptoms emerge, or monitoring patients' responses during treatment.

Published in 2019 [**extraction-of-behavioural-features**], this paper focuses on many channels for collecting user data, with each channel resulting in a set of features. Examples of channels are GPS data, phone usage, step count, etc. The upside of having many channels is improved accuracy since more data is available to describe the behavior of the user, and if certain channels are inactive during certain periods of time, the other channels can cover the missing data, and thus periods of missing data will be avoided with high likelihood.
The paper uses the *AWARE Framework* [**aware2015**] to collect data such as *Bluetooth*, *Call-log*, *Location*, *Campus Map*, *Phone Usage*, *Step Count* and *Sleep State* (via FitBit).

The daily behavior of the user is predicted through four high-level features extracted from the data channels.

User mobility: The movement patterns of the user are derived from location tracking and the campus map feature.

Communication Patterns: Derived from call log and text messages.

Mobility and communication: Derived from Bluetooth device IDs within range, i.e. Bluetooth scan.

Physical activity: Derived from step-count during different windows of the day, i.e. when the user is walking a lot.

The day (24 hours) is split into 4-time windows, which helps identify where the home is since the user likely sleeps the same place most days of the week, which is likely during the *night* time-window.

- Night (00-06)
- Morning (06-12)
- Afternoon (12-18)
- Evening (18-00)

GPS coordinates collected throughout the day have a time stamp connected to them, which makes it possible to calculate a list of interesting features within a day of the user moving around. The features are similar to those found in [**Saeb2015**] adds two additional features, namely *Radius of gyration* and *Time spent at top3 clusters*.
However a couple of additional contributions are found. Firstly a *bout* is defined as a small window of time, which can be defined based on the granularity wanted, i.e. 5 minutes.
Furthermore, a clear definition for the *home* cluster is given as being everything within 10 meters of the center of the home location center. Time spent at home is defined at the time spent within 100 meters of the home centroid. The activity of studying is defined as spending 30 minutes or more in an academic building, while sedentary, which is taking fewer than 10 steps per *bout*.

From the Bluetooth Scan, the devices found each has a unique ID and can, therefore, be categorized into the following groups:

- Self, i.e. the user (the device scanned most often)
- Related, ex-colleagues, flatmates (scanned less often)
- Other, other people (scanned least often)

The three categories are created through K-means clustering, by firstly creating two clusters ($K = 2$), i.e. self and others. Secondly another model is fitted to the data with $K = 3$ clusters, and the model which fits the data the best is chosen, i.e. through cross-validation or BIC.

Phone usage is based on the screen state which at any time can be only one of the following: On, off, lock, unlock. Tracking the state of the screen together with the timestamp at which the state changes, a few features can be calculated, which are:

- Number of unlocks per min
- Time spent interacting with the phone
- Total time unlocked
- Hour of the day of first unlock/turned on
- Hour of the day of last unlock, lock, turned off

- Time spent interacting with the phone (sum of time between unlocking and off/lock)
- Standard deviation of time spent interacting with the phone

By tracking heart rate, and other signals with a FitBit watch, the Fitbit API gives access to the *sleep state* of the user, which at any time is one of the following: Asleep, restless, awake, unknown.
A number of samples will be made during the night and thus the simplest features which can be created are the number of samples in each of the four states. From this, two higher-level features are computed, which are:

Weak Sleep Efficiency: $WSE = \frac{n_{asleep}+n_{restless}}{n_{asleep}+n_{restless}+n_{awake}}$

Strong Sleep Efficiency: $SSE = \frac{n_{asleep}}{n_{asleep}+n_{restless}+n_{awake}}$

With n_{state} being the number of samples for that state.

From the step-count channel features pertaining to the fine-grained activity, without the location, for a whole day, can be calculated:

- Number of steps
- Max steps in a *bout*
- Number of active *bouts*
- Min, Max, Avg. length of sequence active *bouts* in a row

The change in the user's behavior over a longer time period, i.e. a few weeks can be tracked by using a number of the proposed features, and is done by applying a simple linear regression as well as a piecewise linear regression (with two lines) to the dataset. The calculate behavioral change features are then the following:

- Derivatives (a_i) of the linear model: $y = a_1x_1 + a_2x_2 + ... + a_nx_n + c$.
- Change in slope (derivative) for each behavioral feature

Given data for $n \in Z$ timesteps, i.e. weeks, choose some breakpoint $m < n \in Z$ and create the piecewiese linear function:

$$\begin{cases} a_1 x_1 + a_2 x_2 + ... + a_n x_n + c & t < m \\ b_1 x_1 + b_2 x_2 + ... + b_n x_n + d & m \leq n \end{cases}$$

The two-line segments are then fitted to the data with a feature selection algorithm. The resulting slope vectors for the selected features are then behavioral features.

Human Mobility is relevant for a number of different applications such *disease spread, urban planning*, *managing traffic* as well as understanding *social interactions*. Previous contributions within these fields have been using Call Detail Records (CDR) which is the art of triangulating the user's location by use of cell-phone towers. These sources produce very coarse-grained estimates wrt. location and time. For this paper a more traditional location data sampling was used, using the GPS tracker in smartphones, however, for longitudinal studies it can be a problem for the phone battery if the sampling rate is too high. Therefore a more low-energy approach is used where the sampling rate- and the location accuracy are low.

A study with 6 participants was conducted in which location data collected continuously and the users filled out an online diary on a daily basis. Given a series of temporally-ordered data points for a user, the aim was to compute features called *Stops* and *Places of Interest (POI)*. A *POI* is defined as a location of high relevance such as that person's school, gym or a supermarket as is denoted by a type $POI = (ID, lat, lon)$ and a *Stop* at a POI is defined as the period of time in which the user stayed at a POI with a given ID, i.e. $Stop = (arrival, departure, ID)$. To find Stops, both a *Gaussian Mixture Model*, as well as the *DBSCAN* algorithm, were applied for clustering. To evaluate the best model, the f_1 score of each model was computed, and in the end, it was found that both algorithms performed similarly for each of the participants.

In this paper by Canzian et. al from 2015 [**Canzian2015**], it is discussed how existing systems for diagnosing depression require the user to interact with the device. These interactions can be input such as mood-state a few times a day, which can be highly subjective and error-prone. The paper addresses this issue via objective patient data, called *Mobility Features*, which was collected via unobtrusive monitoring by using

smart-phone location data. There was a significant correlation between these *Mobility Features* and depressive moods similar to [**Saeb2015**]. The paper also describes concrete models to *predict* changes in depressive mood by using these features. The paper uses the *PHQ-8* questionnaire as a reference, which is the almost the same questionnaire as the *PHQ-9* (see Appendix **??**), except for only including the first 8 questions. While this paper describes many of the same features as [**Saeb2015**], it does so in a much more detailed and mathematical way, such as with the *Routine Index* feature, which corresponds to the *Circadian Rhythm* feature in [**Saeb2015**].

The feature described is as follows:

Place: $p = (id, t_a, t_d, c)$
A place is a cluster of geo-location points with a unique id and a geographical center $c = (lat, lon)$. When the user arrives at the place it happens at t_a and the user departs from the place at t_d.

Mobility Trace: $MT(t_1, t_2) = [p_1, ..., p_n]$
A mobility trace is a list of all places visited in the time interval (t_1, t_2). $N(t_1, t_2) = n$ is the number of places visited and the time gap between $t_d(p_i)$ and $t_a(p_{i+1})$ is a period of movement.

Total Distance: $D_T(t_1, t_2)$
The total geodesic distance traveled in the time interval (t_1, t_2).

Max Distance: $D_M(t_1, t_2)$
The maximum distance between any two places visited in the interval (t_1, t_2).

Radius of Gyration: $G(t1, t2)$
The geographical area covered in the interval (t_1, t_2)

Standard Deviation of Displacement: σ_{dis}
The standard deviation between each displacement i.e. distances from one point to another.

Home Cluster: H
The cluster most frequented cluster at 02:00, 06:00 and 20:30 on week-days.

Max Distance from Home: $D_H(t_1, t_2)$
The distance to the point the furthest away from home, during the interval (t_1, t_2).

Number of different places visited: $N_{dif}(t_1, t_2)$
The number of different clusters found in the interval (t_1, t_2).

Number of different significant places visited: $N_{sig}(t_1, t_2)$
The number of significant places visited in the interval (t_1, t_2). The maximum number of significant places is 10.

Routine Index: $R(t1, t2)$
The degree to which the place-time distribution during the interval (t_1, t_2) on a given day is similar to the distribution on other days, during the interval (t_1, t_2).

In addition to concrete definitions of features, the paper also provides an algorithm for sampling location in a way which saves phone battery. GPS data collection is very expensive if done with a high frequency and by using cheaper sensors to turn it off when necessary can, therefore, save a lot of battery. Concretely the accelerometer is used by an activity recognition algorithm to determine the user's activity state. Concretely this is done by modelling the user as being in a movement state, of which there are 3: *Static (S)*, *Moving (M)* and *Undecided (U)*, where location tracking only takes place in the U or M states.

The definitions for the states are as follows:

Static (S)
The user is in the same location i.e. school/work. Never sample location data while in this state since it is unnecessary.

Moving (M)
The user is moving from place to place and the phone should, therefore, keep location tracking on, while in this state.

Undecided (U)
It is unknown whether the user is moving or not and moving to either state S or M is now impending. When this state is entered, the user's location l_1 is sampled, and after 5 minutes the location is sampled again, providing l_2. Then, The distance $d = dist(l_1, l_2)$ is calculated and from this, the algorithm will transition to either S or M depending on the distance between the two locations.
The transitions are made, as follows:

U $\rightarrow$ **S**: If the distance $d < 250$ m.

U $\rightarrow$ **M**: If the distance $d \geq 250$ m.

S $\rightarrow$ **U**: If two consecutive activity recognition samples indicate user activity with over 50% confidence.

M $\rightarrow$ **U**: If two consecutive samples are less than 250 m apart, or when an activity recognition sample indicates the user is still with more than 50% confidence.

The prevalence of depression has led to the development of many mobile health-tech solutions and applications for monitoring diseases related to depression. However few of these applications have focused on the treatment of depression. There is clinical evidence showing that depressive symptoms can be alleviated through Behavioural Activation which is a behavior-focused treatment method for treating depression. Rohani is a postdoc at CACHET and has published and developed two systems during his Ph.D. within the realm of Behavioural Activation which is the MORIBUS- and MUBS systems.

The *MORIBUS* system [**moribus**] by Rohani aims to make it easy for depressed patients to remember and register their behavior in order to increase their every-day positive outcomes. The system comes with a catalog of 54 pre-made activities such as *Get ready in the morning* and *Eat breakfast* and has support for patients to add their own activities by manually inputting text and a rating. When an activity is completed, the patient registers this in the app along with a short reflection. The application is able to provide the user with statistics and summaries of activity frequency and reflections.

The *MORIBUS* system later evolved into the *MUBS* system which was rewritten in Flutter [**mubs-rohani**]. The application uses a larger catalog of pleasant activities and gives recommended activities to perform, personalized to each user. The catalog is a database of 384 common enjoyable activities where each activity has been manually labeled by researchers with an *activity level*, as well as a *category*. The recommender model uses a Naive Bayes model with a set of features representing each feature and a corresponding label representing whether it is positive or negative for that individual user. The model is designed to recommend already-known activities as well as novel activities to the user which is in the spirit of behavioral activation. Future work of

the system includes adding additional features such as *Location*, *Local Weather* and *Ambient Noise* to provide an even more detailed context to the system.

Both systems rely on manual input from the user in order to learn which activities to recommend in the future. This can pose a problem, especially when running longitudinal studies since participant retention may suffer if the input process is too involved for the user. These systems by Rohani, therefore, provide examples of use cases for the automated feature generation algorithm such as the *Mobility Features Package* which is easy to integrate into an existing Flutter application.

The *AWARE Framework* [**aware2015**] is an open-source toolkit and a reusable platform for capturing, inferring, and generating user-contexts on mobile devices. Phones possess high-quality sensors but are resource-constrained with regards to their processing speed and battery capacity, which must be considered when computing contexts in real-time. The *AWARE Framework* therefore aims to ensure an easy way of collection contexts for the application developer, and it is demonstrated how the tool can reduce the software development effort of researchers when building mobile tools for developing context-aware apps and doing so with minimal battery impact. By designing an API that conceals the underlying implementation of sensor data-retrieval, and exposes an abstract representation of a *user context object*, AWARE shifts the focus from software development to data collection and analysis. Currently AWARE is available on both iOS and Android and supports a number of data channels [1] such as the built-in sensors, as well as *Application Usage*, *SMS*, and *Phone Call Logs*. Some of the channels are however not available on iOS due to the iOS developer API, in general, being more restrictive than that of its Android counterpart.

The CARP Mobile Sensing (CAMS) Framework[2] is a Flutter framework for adding digital phenotyping capabilities to a mobile-health app. CAMS is designed to collect research-quality sensor data from the many smartphone data channels such as sensors and location data, in addition to external sensors that the phone is connected to, such as wearable devices. The main focus of the framework is to allow application programmers to design and implement a custom mobile health app without having to

[1] `https://awareframework.com/sensors/`
[2] `https://github.com/cph-cachet/carp.sensing-flutter/wiki`

start from scratch, with regards to the sensor integration, by enabling the programmers to add an array of mobile sensing capabilities in a flexible and simple manner. This would include, firstly, adding support for collecting health-related data channels such as *ECG*, *GPS*, *Sleep*, *Activity*, *Step Count* and much more. Secondly, to format the resulting data according to standardized health data formats (like *Open mHealth* schemas [3]. Thirdly, to upload the collected data to a specific server, using a specific API (such as *REST*), in a specific format, such that it may be used for data analysis. To include as many data channels as possible the application should also be able to support different wearable devices for ECG monitoring and activity tracking. Hence, the focus is on software engineering support in terms of a solid programming API and a runtime execution environment, that is being maintained as the underlying mobile phone operating systems and APIs are evolving. Moreover, the focus is on providing an extensible API and runtime environment, which together allow for adding application-specific data sampling, wearable devices, data formatting, data management, and data uploading functionality to an application. In addition, in order to reduce the number of integrations needed, the framework must be cross-platform such that it supports both Android and iOS, without having a codebase for each platform.

The data collection pipeline is defined firstly by a *Study* which contains all the information needed to conduct a real-world study, which includes the collection of certain data types with a set of rules and saving it somewhere.

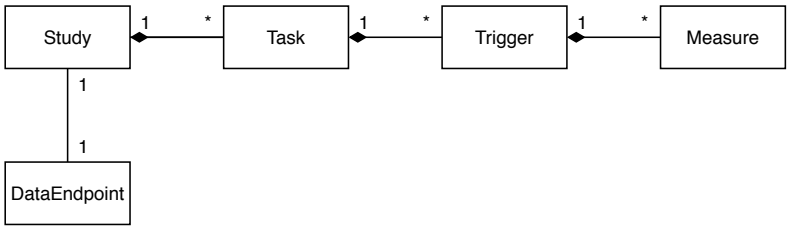

Figure 2.3: A UML diagram of the *CAMS* Domain Model..

Concretely, a *Study* holds a set of *Triggers*, which define *when* sampling is carried out, such as by scheduling sampling at given time each day, or when a certain event is registered, such as entering a specific *GeoFence*[4].
Each *Trigger* holds one or more *Tasks*, each of which define which *Measures* should run simultaneously. In addition, a *Task* also defines whether or not data is sampled in the background, or whether the user needs to interact with the device in order to perform sampling.

[3] `https://www.openmhealth.org/documentation/#/schema-docs/schema-library`
[4] `https://developers.google.com/location-context/geofencing`

Each *Task* holds a set of *Measures* each of which defines what to sample, i.e. which data channel to listen to. Lastly, a *Study* also holds a reference to the *DataEndpoint* specifying which where data ends up, i.e. by uploading it to a server.
An integration for the CAMS Framework will be described later in this thesis.

This chapter will discuss, the requirements for the software package. Firstly, a brief overview of the technology used will be given, this includes GPS data and an architecture overview. Next it will be discussed which features should be computed to produce a useful mobility context, and how some of these algorithms can be designed to work in real-time. Thirdly, the underlying data model design is discussed which has implications for how the information flow of the implementation looks like. Lastly the philosophy of design of the public-facing API will be discussed.

The Mobility Features which were used were a subset of the features discussed by [**Saeb2015**; **Canzian2015**]. In addition a set of, what we will refer to as *intermediate features* were also used which from the work of [**sparse-location-2014**].

Common for many of the algorithms for finding user mobility features is that they rely on clustering of data points, in order to find the number of Places. However when dealing with large amounts of data points it may be necessary to reduce the initial amount of data points such that these clustering algorithms are able to run faster. This downsampling process will be carried out by clustering raw data points into what we shall refer to as *Stops* indicating locations where the participant did not move around a lot. The *Stop* notion is loosely based on the [**sparse-location-2014**]. The pre-processing produces *intermediate features* from which the final mobility features are derived. These intermediate features are *Stops*, *Places* and *Moves* and provide a coarse-grained version of the dataset which makes the final feature calculation much cheaper, computationally speaking. We define *Places* as specific locations of relevance to the user, such as home or workplace. *Stops* are specific visits to any of those places. Thus, a *Stop* is always associated with a single *Place* while places can be associated with one or more *Stops*. Finally, *Moves* are the sequences of location samples in between *Stops*, representing moving between *Places*.

The features derived from this class are *Home Stay*, *Location Variance*, *Number of Places, Entropy*, *Normalized Entropy* , *Distance Travelled* and *Routine Index*.

Features used Why tthose features what do they mean

Coming up with a domain model is a process that can provide clarity and direction for a software system, even on a small scale such as in a library.

In order to capture the data model in an object-oriented programming language, a UML diagram was maintained as the implementation went along in order to keep track of relationships between the classes.

A *Location* is defined by a geographical *latitude* and *longitude* and represents a real-life location.

A *Cluster* is created from a collection of *Locations*. This class serves as an auxiliary class for clustering based on median *latitude* and *longitude*.

A *Single Location Data Point* (SLP) is a time-stamped *Location*. By having a time-stamp, a collection of SLPs may be ordered and grouped by the time of day. In essence, the SLP is a Data Transfer Object (DTO) [1] which is used to transfer GPS data from an arbitrary Location plugin to the *Mobility Features Package*.

An *Hour Matrix* is a matrix with 24 rows and columns equal to the number of places of some period. The *Hour Matrix* class is used to calculate the *Routine Index* feature, as well as to identify the *Home Cluster*, which is the place most visited during 00:00 and 06:00. An Hour Matrix is constructed from a list of *Stops* which all have the same date.

[1] `https://martinfowler.com/eaaCatalog/dataTransferObject.html`

A *Stop* is a visit at a known Place (see below) for an extended period of time. A *Stop* is defined by a location that represents the centroid of a collection of data points, from which a *Stop* is created. In addition a *Stop* also has an *arrival-* and a *departure* time-stamp, representing when the user arrived at the place and when the user left the place. From the arrival- and departure timestamps of the *Stop* the duration can be computed.

A *Place* defined as a group of stops that were clustered by the DBSCAN algorithm [**density-based-1996**]. From the cluster of stops, the centroid of the stops can be found, i.e. the center location. In addition, it can be computed how long a user has visited a given place by summing over the duration of all the stops at that place.

A *Move* is the displacement of the user from stopping s_a to stop s_b in which the user passes through a series of SLPs. Given the distance traveled from stop s_a to stop s_b, in addition to the departure of s_a and the arrival at s_a the average speed at which the user traveled can be derived.

A *Mobility Context* is a collection of features which are derived from a set of intermediate features, where the *Stops* and *Moves* are from a specific date. The *Places* is derived from multiple dates for reasons which will be explained in the implementation details. In addition, a set of *Mobility Contexts* from previous dates can be provided as an optional parameter. A Mobility Context contains the mobility features, although the *Routine Index* is only available if an array of the set of *Mobility Contexts* was provided as a parameter, which is due to the feature depending on the data from previous days in order to compare them.

Here, an overview of the algorithms used by the *Mobility Features Package* will be provided. The overview will not discuss implementation details but will provide precise definitions for how these are computed and the considerations which were made. Most of the features were simple to implement with support for real-time computation and were just a matter of performing arithmetic with regards to distance and time spent and places, however, the ones which need some explanation are outlined in this section.

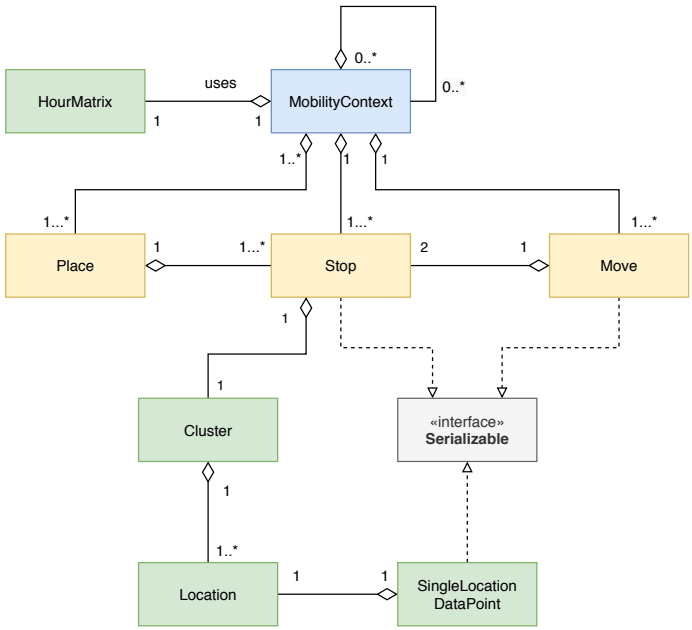

Figure 3.1: UML diagram for the classes used in the *Mobility Features Package*.

A period is a set of several dates is defined as $D = \{d_1, d_2, ..., d_{|D|}\}$ with $|D| \leq 28$ and the date following the period being d_t. This can also be translated as D are the historical dates to the date d_t.

A *Single Location Point* is a timestamped location and is defined by the tuple $x = (T, l)$ where T is the exact timestamp and l is the Location defined as a geographical point on the globe. The distance between two *Single Location Points* is defined as $\delta(x_a, x_b) = \delta(l_a, l_b)$ and δ is the *Haversine* distance function.

Finding *Stops* is done by traversing every *Single Location Points* in temporal order, i.e. the timestamp is used. The *Stops* for a given date is found by clustering Single Location Points on that date based on time and distance.

The set of Stops found for the period D is defined as

$$S = \{s_1, s_2, ..., s_{|S|}\} \mid s_i = (T_{arr}, T_{dep}, l)$$

The triple (T_{arr}, T_{dep}, l) denotes the arrival timestamp, the departure timestamp and the cluster location for the Stop s_i, respectively.

Places can now be found by applying the *DBSCAN* algorithm to the *Stops* found. It is important to note that if the *Routine Index* for a given date over a given period is to be evaluated later on, all *Stops* found for this period should be used. This means all *Stops* from previous dates need to be stored on the device.

The set of Places found for the period D is defined as

$$P = p_1, p_2, ..., p_N \mid p_i = \{s_1, s_2, ..., s_{|p_i|}\}$$

Moves for a given can be calculated using the *Stops*- and the *Single Location Points* from that date. The *Moves* are found by going through each *Stop* and calculating the distance between the current *Stop* and the following *Stop* by going through all the *Single Location Points* which were sampled in the time interval between these the *Stops*. These points form the path which was taken between the two *Stops* and the path is used to calculate the exact distance traveled.

A set of*Moves* is defined as

$$M = \{m_1, m_2, ..., m_{|M|}\} \mid m_i = (s_a, s_b, X_i), X_i = \{x_1, x_2, ..., x_{|X_i|}\}$$

is a set of time-ordered *SLPs*.

This matrix is made from all the *Stops* on a given day, each of which belong to certain *place* and has an *arrival* and *departure* timestamp. From this it can be calculated exactly which hour slot(s) to fill out and the duration to fill that slot with. For simplicity, we define a couple of constraints on the *Hour Matrix*:

- The *Hour Matrix* has exactly 24 rows, each representing 1 hour in a day.
- The number of columns represents the number of *Places* for the period.

- An entry represents the portion of the given hour-slot that was spent at a given *Place*.
- Each row can maximally sum to 1.

Formally, given a period for which the number of *Places* is given as N the *Hour Matrix* H for a given day d is defined as:

$$\mathsf{H}(d) \in [0,1]^{24\times N}, \sum_{j=1}^{N} \mathsf{H}^{d}_{i,j} \leq 1$$

Given an array of *Hour Matrices* for a period with D days, the *Mean Hour Matrix* for the period is defined as the average entry for each *Hour Matrix* in the period which are indexed with the date d_k:

$$\mathsf{H}^{\mu}(D)_{i,j} = \sum_{k=1}^{|D|} \frac{1}{|D|} \mathsf{H}(d_k)_{i,j}$$

Given a Stop s with arrival T_{arr} and departure T_{dep} the hour matrix as follows:
Let $i = hour(T_{arr})$, $j = hour(T_{dep})$ and $\Delta_T(\cdot)$ is the function for calculating the duration in hours.

$$\mathsf{H}_{i,p} \leftarrow \mathsf{H}_{i,p} + T$$

.
If $i = j$ then $T = \Delta_T(T_{dep} - T_{arr})$, otherwise the following algorithm is applied:

$$\mathsf{H}_{i,p} = 1 - T$$

Where the value of T depends on the variable $k = i$ up to j:

	Place #1	**Place #2**	**...**	**Place #N**
00 - 01				
01 - 02				
...				
16 - 17				
17 - 18				
18 - 19				
...				
23 - 00				

Figure 3.2: An *Hour Matrix*.

$$T(k) = \begin{cases} \Delta_T(T_{dep} - T_{arr}) & \text{if } i = k = j \\ 1 - (hour(T_{arr}) - T_{arr}) & \text{if } i = k < j \\ 1 & \text{if } i < k < j \\ hour(T_{arr}) - T_{arr} & \text{if } i < k = j \end{cases}$$

The Home Stay feature indicates the percentage of time spent at the Home cluster, out of all the time of the day. Firstly a definition for Home needs to be clear; in the literature it is defined as the cluster which the user on average spends the most time at between 00:00 and 06:00 over a period of time [**Saeb2015**; **Canzian2015**]. However since the *Home Place,* may change from day to day, it was decided to let *Home* for a specific day be defined as the cluster for which the most time was spent during 00:00 and 06:00 on that day only. Formally, the *Home Place* $p_h(d_t)$ for today d_t is the place p_h where the following holds:

$$h = \underset{n}{\operatorname{argmax}} \sum_{m=1}^{6} \sum_{n=1}^{N} \mathsf{H}(d_t)_{m,n}$$

However in the literature [**Saeb2015**; **Canzian2015**] it is not stated how this would be calculated for an incomplete day. A choice was made for this to be calculated using the sum of durations for all *Stops* belonging to today's *Home Cluster*, divided by the time elapsed since midnight. The duration *Stop s* will be denoted $\Delta T(s)$ and similar the duration spent at a place p is defined as $\Delta T(p)$

$$\Delta T(p_h(d_t)) = \frac{\sum_i \Delta_T(s_i) \mid s_i \in p_h(d_t)}{T_{now} - T_0}$$

Where $T_{now} - T_0$ is the time elapsed since 00:00:00.

First we define the distance of a Move m_i as the sum of all the distances in the 'chain' of points in X_i, i.e.

$$\delta(m_i) = \sum_{j=1}^{|X_i|-1} \delta(x_j, x_{j+1})$$

The *Total Distance Travelled* for a date d_t is defined as

$$\delta(d_t) = \sum_{i=1}^{|M|} \delta(m_i)$$

where M refers to the moves on date d_t.

Within the field of Information Theory, entropy is described in [**information-theory**] as a quantity associated with a random variable, and can be interpreted as the average level of *information* contained within the outcomes of that variable.
The Entropy $H(X)$ of the set $X = \{x_1, x_2, ..., x_n\}$ is defined as

$$E(X) = -\sum_{i=1}^{n} p(x) \log p(x)$$

.
The Entropy is maximised if $p(x) = \frac{1}{n}$ for all $x \in X$, i.e. all outcomes are equally likely. If this is the case, the Maximum Entropy becomes

$$H_{max}(X) = -\sum_{i=1}^{n} p(x) \log p(x) = -\sum_{i=1}^{n} \frac{1}{n} \log \frac{1}{n} = -n\frac{1}{n} \cdot -\log n = \log n$$

We define the Normalized Entropy as the Entropy H divided by the maximum possible entropy H_{max}:

$$H_N(X) = \frac{H(X)}{H_{max}(X)} \in [0, 1]$$

The Normalized Entropy (NE) makes it easier to compare Entropy values of different distributions since they all reside on the same scale, being a scalar value between 0 and 1. An NE value near 1 indicates that the X follows a uniform distribution where every outcome is equally likely. A small NE value indicates that the distribution is very skewed, with certain outcomes having very high likelihood and some having much lower likelihood.
In the context of user mobility, we can view the time user spends at a certain place as the outcome of the place variable, i.e.

$$H(P) = -\sum_{i=1}^{n} Pr(p_i) \log Pr(p_i)$$

and

$$H_N(P) = \frac{H(P)}{H_{max}(P)} \in [0, 1]$$

where P is the set of places visited today and $Pr(p_i)$ refers to the time spent at place p_i today. It must holds that such that $Pr(p) > 0$ for $p \in P$, otherwise the term $\log P(p_i)$ cannot be evaluated since $\log(0)$ is undefined.. The concept of NE gives us a tool to say something about where the user spends their time; a high NE value indicates they spend their time uniformly among the places, whereas a low value indicates that the user spends most of their time at very few places.

The features described in the literature by [**Saeb2015**] which can be found in Section **??**. The time distribution for a day is defined by spending a duration of time at a certain space within a certain time-slot. However the *Routine Index* [**Saeb2015**; **Canzian2015**] was much more demanding to implement and it will, therefore, be a feature that is highly discussed in this thesis. To recap, the *Routine Index* describes how similar the place-time distribution of a given day is, compared to previous days for some period of days. A concrete period length of 28 days was chosen, which means the *Routine Index* of today describes how similar today was to each day during the last month. However the implementation by [**Canzian2015**] was very complex and therefore a much more simple version was chosen for the first iteration of the software package. We define the *Routine Index* as a similarity measure with a value between 0 and 1, where a low value indicates little overlap and a high value indicates a high degree of overlap. By representing each day with an Hour Matrix the similarity function can be defined.
Lastly, we define a union operator $A \cap B$ defining the *overlap* of two matrices A and B as

$$A \cap B = \sum_{i=1}^{24} \sum_{j=1}^{N} \min(A_{ij}, B_{ij}) \mid A_{ij} > 0, B_{ij} > 0$$

The *Routine Index* for today d_t given the historical dates D, is defined as:

$$r(d_t, D) = \frac{\sum(\mathsf{H}(d_t) \cap \mathsf{H}^{\mu}(D))}{\min\Big(\sum \mathsf{H}(d_t), \sum \mathsf{H}^{\mu}(D)\Big)}$$

The numerator

$$\sum(\mathsf{H}(d_t) \cap \mathsf{H}^{\mu}(D))$$

defines the *actual overlap* as the sum of the overlapping entries between today's data and the historical data. The denominator

$$\min\Big(\sum \mathsf{H}(d_t), \sum \mathsf{H}^{\mu}(D)\Big)$$

is the smallest sum of either matrix, whichever is smaller, and defines the *maximum potential overlap* between the two matrices. If one matrix is very sparse then the potential overlap is very low, and vice versa. If the *actual overlap* and the *maximum potential overlap* are the same, it means the matrices are the same and the Routine Index will have a value of 1.

If the features have to be evaluated at any point of the day, as is the case for real-time computation then the Routine Index cannot rely on a full day of data. To make

the feature represent something meaningful in real-time it would have to reflect the routine of the user up until the current time of days, i.e. if calculated at 14:00 then it should only use the first 14 rows of the matrix. This means the *Routine Index* may be high early in the day since people usually sleep the same place, but are open to deviating as the day progresses. This can be useful to an application programmer in a recommender-system setting, where a trigger based on the *Routine Index* could be set, such that the user could be alerted when the value falls below a certain threshold.

Most people will go on vacation during the year, which means the place where they sleep changes. In general, peoples' habits will inevitably change somewhat over time, and if one compares the routine of a certain person now to what their routine looked like a year ago, it is not unlikely to be very different. However just because a user changes their routine over time, does not mean they don't currently possess one. Therefore it was chosen to base the *Routine Index* was chosen to be calculated based on the last 4 weeks of data in order to base the routine overlap on more recent days. An issue which was not dealt with is the fact that the routine on weekdays differs a lot from the routine during the weekend. This is especially true for people who spent 8 or more hours at work during the weekdays and spent those 8 hours somewhere else during Saturday and Sunday since it means the *Routine Index* cannot exceed $\frac{2}{3}$ due to a third of the day's total hours being spent at a different place than usual. To add to this, even weekdays may look slightly different from one another, especially for those who are part of sports clubs which meet during certain days of the week. In future work it would be interesting to investigate whether or not comparing Mondays to Mondays and vice versa for every day in the week would yield more accurate results.

In order to calculate the *Routine Index* we need to save and load the historical data somehow. Two approaches were considered:

The first approach involves keeping historical stops saved on disk. By doing so, the *Places* can be found by clustering the stops with DBSCAN, and an Hour Matrix for each day in the last 4 weeks can then be computed and the average Hour Matrix can be derived from these. Afterward the *Routine Index* can be calculated as the overlap between these two, as previously described. In the field study, the author had just over 70 stops per week, which corresponds to around 300 stops for a 4 week period, which is a very manageable number of elements to cluster with DBSCAN, which would be the only bottleneck in this approach.

The second approach involves keeping storing the Hour Matrix for each day on disk. However for this approach the *Places* would also need to be saved such that places collected today could be compared (wrt. distance) to historical places. The reason

this is necessary is to ensure that the matrix dimensions of all the matrices are the same and that these columns refer to the same *Place*. This approach has the potential to be computationally cheaper but is much more complex to implement and was therefore not chosen in this iteration.

Location data sampled on a mobile device contains information such as the geospatial position on the globe, as well as a time-stamp. Location data may also include information such as altitude, accuracy, and speed, but these attributes were not considered. Tracking location data on a smartphone on Android or iOS works by calling a Location API that allows the streaming of location data. The location data should be tracked in the background since the user cannot be expected to not use their phone at all during the time they are being tracked since this is virtual all the time. For this to work, certain permissions and flags must be configured correctly on the respective platforms such that the application using this package can sample location data and process it in the background. Shortly after the start of this thesis, Google limited how the Android Location API handles background updates, which meant that an Android application must be in the foreground to track location such as is the case with Google Maps navigation where it is very explicit[2]. Due to time constraints however, a good solution to this problem was not found.

The Flutter Framework is a cross-platform development framework that was released by Google in 2018. Flutter uses the Dart programming language and makes it possible to write a codebase purely in Dart, and compiling this code to native iOS and Android code. What this means is that an application developer can create an application for both platforms with a single codebase. However whenever native-specific APIs have to be invoked such as that of the camera or various phone-sensors, a package developer has to write a library that does so, by means of a method invocation from the Dart language which triggers the corresponding API invocation ob the Android or iOS platform. This method invocation library is referred to as a *plugin* within the Flutter world, in contrast to a *package* which simply invokes other Dart code and as such contains no platform-specific source code. The *Mobility Features Package* will be a Flutter package that contains a collection of algorithms that provides an application programmer with object-oriented abstractions that allows him/her to calculate relevant features for a mobile health application. The Location API, available on both iOS and Android, will not be invoked directly from this package since that

[2] `https://developer.android.com/training/location/background`

would require it to be a plugin. This has two main upsides: From the point of the application developer, it allows him/her to use their location plugin of choice (of which there are many [3] with specific parameters for how the location is tracked (ex frequency and distance). Secondly, from the perspective of the maintainer of this package, the package becomes much more modular and in turn easier to maintain.

The software package is intended to be used by a Flutter developer and has to have an *Application Programming Interface* (API) that strikes the balance between being easy to use and hiding complexity from the programmer. This can be done in numerous ways, but given that Flutter uses the object-oriented programming language, Dart, it makes sense to encapsulate complexity using Classes rather than pure functions. With programming languages such as Python which are not inherently object-oriented, the APIs often expose a series of functions that return basic objects such as tuples of floats, integers, and strings, rather than objects to the application programmer.

Location data is very sensitive and with a very high sampling rate provides a high level of insight, down to the second in fact, of where the user was during the day. For this reason, it is the aim of the Mobility Features Package to not let the location data 'leave the phone' so to speak. Instead, only a set of *mobility features* that contain less sensitive information can be outputted. However, since the *Stops* need to be saved manually on the device, it was chosen to give access to pre-processing features such as *Stops*, *Moves* and *Places* since they are also needed to make *Mobility Context*. It may also be the case that they are very useful on their own for compressing raw GPS data into a much smaller and more coarse-grained dataset that captures where the user was stationary. This can be very useful if combined with reverse geocoding in which the geo-locations of *Places* are converted to addresses, which can then be further transformed into a place category such as school, work, grocery store, sports club and much more, which can give insights into what the user is actually spending their time on. Reverse geocoding will however not be part of this thesis. The intermediate features do however contain very sensitive information, and probably more sensitive that raw GPS data because the exact places the user was at and moved between are stored, and the noisy data is removed. In a future iteration these intermediate features will likely either be contained in their own Flutter package, which the Mobility Features Package depends upon and then hidden away in the Mobility Features Package such that not sensitive information may escape the package.

[3] `https://pub.dev/packages?q=location`

The main output to the application developer will be a list *Mobility Context* objects which have the described mobility features as fields. The *Mobility Context* class takes in Date, as well as a List of Stops, Moves, and Places, where the Stops and Moves are on the given Date, and the Places are from a given period, i.e. from today and as far back as has been tracked with the maximum being 28 days prior.
Rather than using both Places and Stops for instantiation, the data model could be modified such that a Place contained a list of Stops made at that place. This would mean grouping the Stops by Place rather than Date, and would require filtering to take place every time a *Mobility Context* object is created, to remove all Stops, not on that specific date. In a real-world scenario the application developer will usually have the SLPs for the current day available, and from those the Stops today can be generated which means no filtering is required. Grouping Stops into places would however lead to a nicer data model, but a design choice was made in favor of less computation.

It was chosen to include a serialization API in the package since the algorithms rely on using historical data to compute the Routine Index. Otherwise the programmer would have to save this themselves, which is not very user friendly. While the serialization interface was provided in order to easily support the saving and loading of SLPs, Stops, and Moves it is however not automatic. It was left up to the application programmer to perform this since it is only necessary if the *Routine Index*is desired and will make the data collection and computation pipeline much more complex. However in future work it would make sense to use a real database on the phone instead, which supports querying objects using their Date field.

CACHET Copenhagen Center for Health Technology

MDD Major Depressive Disorder

BA Behavioural Activation

Major Depressive Disorder (MDD)
Also known simply as depression, is a mental disorder characterized by low mood, low self-esteem, loss of interest in normally enjoyable activities, low energy, and pain without a clear cause.

Behavioural Activation (BA)
A therapeutic intervention that is often used to treat MDD. BA treatment focuses on keeping depression at bay through activities which produce positive reinforcement for the patient. BA is highly customizable and is a very personal treatment plan.

Mobility
How a person changes location over time.

Mobility Feature
A number which describes one of the ways in which a person changes location over time. Many features will paint a more complete picture of the mobility patterns than fewer features.

Pre-processing
TODO.

Patented by Pfizer
The PHQ-9 questionnaire contains 9 questions pertaining to the mental state of the patient [1]. Each question asks 'Over the last two weeks, how often have you been bothered by any of the following problems?' with the questions being the following:

Q1: Little interest or pleasure in doing things?

Q2: Feeling down, depressed, or hopeless?

Q3: Trouble falling or staying asleep, or sleeping too much?

Q4: Feeling tired or having little energy?

Q5: Poor appetite or overeating?

Q6: Feeling bad about yourself, or that you are a failure, or have let yourself or your family down?

Q7: Trouble concentrating on things, such as reading the newspaper or watching television?

Q8: Moving or speaking so slowly that other people could have noticed. Or the opposite – being so fidgety or restless that you have been moving around a lot more than usual?

Q9: Thoughts that you would be better off dead, or of hurting yourself in some way?

Each question can be answered with the following 4 possibilities, each giving a number of points indicated in brackets:

- Not at all (0 points)
- Several days (1 point)
- More than half the days (2 points)

[1] `https://patient.info/doctor/patient-health-questionnaire-phq-9`

- Nearly every day (3 points)

At the end of the survey, the points are summed up and the patient is categorized into one of 5 categories based on the number of points acquired:

- Less than 5 (no depression)
- 5-9 (mild depression)
- 10-14 (moderate depression)
- 15-10 (moderate/severe depression)
- Greater than 20 (severe depression)

The EMA Questionnaire contains six short questions each targeting a common mental state strongly related to depression, each for which the user has to give a self-perceiving rating from 1 (none) to 7 (extreme):

- Negative Affect
- Hopelessness
- Anhedonia (loss of interest)
- Fatigue/Energy
- Loneliness
- Positive Affect

{2.7182818284}ϑφερτυθιοπσδφγηξκλ