QROWD - Because Big Data Integration is Humanly Possible

Innovation action

D6.3 – Spatio-Temporal Analytics

<table>
<thead>
<tr>
<th>Author/s</th>
<th>Patrick Westphal (InFAI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reviewer</td>
<td>Pauline Baudens (TomTom)</td>
</tr>
<tr>
<td>Due date</td>
<td>31.05.2019</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>PU</td>
</tr>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
</tbody>
</table>
Table of contents

ABSTRACT 3

EXECUTIVE SUMMARY 4

1. INTRODUCTION 5

2. HUMAN-MACHINE WORKFLOW FOR MODAL SPLIT 5

3. TEMPORAL ANALYTICS 7
   3.1 Data sources 7
   3.2 Feature Extraction 7
   3.3 Smoothing 11
   3.4 Classifiers 13
   3.5 Results 13

4. SPATIAL ANALYTICS 15
   4.1 Spatial Reasoning 16
   4.2 Spatial Concept Learning 17

5. CROWD FEEDBACK 18

6. CONCLUSIONS 19

REFERENCES 21
LIST OF FIGURES

Figure 1: Analytics workflow for the mode detection task
Figure 2: Example accelerometer reading during a bus ride
Figure 3: Accelerometer reading split up into consecutive, overlapping windows
Figure 4: SHAP values of the considered features computed for an XGBoost model
Figure 5: Comparison of the effectiveness of different smoothing techniques
Figure 6: Average accuracy of leave-one-user-out cross validation for different window sizes and machine learning algorithms
Figure 7: Overview of the overall spatial learning architecture
Figure 8: Bike ride trajectory exemplifying the spatial relation 'runs along'
Figure 9: Example screenshots of feedback channels though the i-Log mobile app
ABSTRACT

Citizen sensing provides means to capture valuable information to improve the mobility infrastructure of a city in many ways. The ubiquity of smartphones as well as their rich sensor support makes them a promising source for citizen sensing. In this deliverable we show how two particular sensors, the acceleration and GPS sensor, can be used to get an overview of participants’ commute habits through spatial and temporal analytics. The insight gained from these analytics techniques may be aggregated to give an overview of the portions of particular types of transportation used in a certain area, called modal split. We examine different machine learning methods and show how they can be applied to the sensor data captured on a citizen’s smartphone. We further give an overview how these techniques could be improved by allowing the citizen to give feedback on the analytics results.
EXECUTIVE SUMMARY

This deliverable describes the spatial and temporal analytics capabilities of the QROWD analytics component. We showcase the analytics techniques developed for the use case BC2-UC#1 ‘Modal Split’ where the main goal is to detect a citizen’s transportation modes given sensor data captured on the citizen’s mobile phone.

The deliverable serves as a technical documentation for developers involved in implementing the analytics component of the QROWD infrastructure, and for persons contributing to other work packages sharing interfaces to this component. Besides this, it specifies the overall approach of how we are tackling the problem of deriving transportation modes from sensor data which could be of interest to other researchers.

In particular we share our insights and evaluation results we collected during the implementation of the spatial and temporal analytics methods, as well as data preprocessing steps that were necessary for the use case at hand. In this regard this deliverable is an extension of the deliverable D6.1 which covered the overall architecture of the QROWD analytics component. We further present crowd feedback methods which can be incorporated into the process to improve the performance of the transportation mode prediction.

The main outputs of this deliverable are the respective implementations for spatial and temporal analytics techniques, as well as for the data preprocessing. We further provide evaluation results to give an impression of how well the developed techniques perform on the BC2-UC#1 ‘Modal Split’ use case.
1. INTRODUCTION

In this deliverable we give an overview of the analytics components aiming at finding discriminative spatial and temporal patterns in citizen sensing data, also incorporating citizen feedback. Here we focus on the use case BC2-UC#1 ‘Modal Split’ as described in the deliverable D2.2. For this use case citizens of the city of Trento provide sensor data via the mobile app i-Log on a daily basis (see deliverable D2.4 for more details). The main analytics task here is to learn classifiers that can infer the transportation modes used by these citizens while commuting. The modes to distinguish are car, motorcycle, train, bus, cableway, bike and walk. Based on the detected modes one can derive the overall modal split.

The two main data sources available on a citizen’s smart phone, and considered for this task are:

- the acceleration sensor, or accelerometer (capturing accelerations for each dimension in space, together with the timestamp when the acceleration sample was measured)
- GPS sensor (capturing GPS positions together with attached timestamps when each of these GPS position was measured)

Thus, the possible attempts for finding discriminative patterns in these data are:

- a) temporal patterns in accelerometer data (e.g. two acceleration peaks per second, one for each step taken, if a citizen was walking)
- b) temporal patterns in GPS trajectories (e.g. the trajectory started at the departure time of a certain bus line, if a citizen commuted by bus)
- c) spatial patterns in GPS trajectories (the commute started at a bus station, if the citizen commuted by bus).

We realized that to establish attempt b) we would need very accurate GPS trajectories, which contradicts with the battery constraints we have. There is a trade off between having a good temporal resolution of a GPS trajectory and keeping the impact on a citizen’s mobile phone battery low. Since the user experience w.r.t. to the citizen sensing mobile app i-Log was more important here, we did not pursue attempt b) but focused on a) and c).

Besides making available sensor data, the mobile app i-Log also provides a channel back to the citizen we collected the data from. This opens up the option of getting feedback w.r.t. the transportation modes we detected. On the one hand side this allows getting labeled data for bootstrapping the machine learning, on the other hand side this might guide the machine learning algorithms in cases where they can only make predictions (i.e. detect a transportation mode) with low confidence.
2. HUMAN-MACHINE WORKFLOW FOR MODAL SPLIT

To solve the use case BC2-UC#1 ‘Modal Split’ we came up with a workflow involving results of

- WP3 for establishing an efficient and effective feedback channel to the citizen via the i-Log mobile app (c.f D3.2) (referred to as ‘crowd services’ in the following)
- WP7 to access citizen sensing data (cf. D7.3) (referred to as ‘raw i-Log data’ in the following)

Figure 1 provides a WP6 centric view on this human-machine workflow for modal split.

![Analytics workflow for the mode detection task](image)

Fig. 1: Analytics workflow for the mode detection task

Within the overall workflow one can distinguish two main parts: Those steps and components involved in training, or ‘learning’, transportation mode classifiers, and those involved in doing the actual transportation mode predictions. Moreover, there are preprocessing steps required. Given a full day sensor reading we first have to detect the commute parts, i.e. extracting those movements that may contribute to the modal split. Further we have to extract features the machine learning algorithms can work on. Both pre-processing steps are the same for the learning and prediction part.

To train a classifier, first the labeled raw i-Log data, i.e. the accelerometer and GPS sensor readings, are pre-processed. This step is discussed in more detail in Section 3.2. For the actual training phase we distinguish those machine learning algorithms that work on the streams of acceleration values, named numeric machine learning (ML) approaches, and those learning discriminative patterns from spatial relations, termed symbolic ML approaches. Those two classes of algorithms also exactly cover the two different pattern learning attempts introduced in Section 1: The numeric ML algorithms are trained to find temporal patterns in accelerometer data, whereas the symbolic ML algorithms shall learn distinctive spatial patterns in GPS trajectories.

To run the actual transportation mode prediction, again pre-processing steps are
Spatio-temporal Analytics

performed. Afterwards all trained classifiers will make an individual prediction and an overall prediction is derived, e.g. based on a majority vote and the algorithms’ confidences.
3. TEMPORAL ANALYTICS

3.1 Data sources
For temporal analytics we focus on learning classifiers which can detect the transportation mode based on accelerometer sensor readings. A short segment of such sensor data is shown in Figure 2.

![Example accelerometer reading](image)

Fig. 2: Example accelerometer reading during a bus ride for a captured time period from roughly 13:30:00 to 13:32:15 with its x (green), y (red) and z (blue) components

One accelerometer data point comprises three components, each expressing the acceleration of one dimension in space. The data stems from Android devices and was captured via the Android sensors API which allows suggesting different sampling periods on a change listener. However, these sampling periods are just suggestions and the actual sampling frequency depends on the system load etc. [1] which might cause variations in the sampling frequencies and even gaps in the overall accelerometer data stream.

3.2 Feature Extraction
Based on the three acceleration components of an accelerometer sensor reading there is a variety of actual features that were proposed in, or can be derived from the literature (e.g. [2-7]):

F1) the magnitude of a data point, i.e. the square root of the sum of the squares of the x, y and z components $\sqrt{x^2 + y^2 + z^2}$

F2) the minimum w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F3) the maximum w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F4) the mean w.r.t. a certain component (or the magnitude) of a
considered window of consecutive data points

F5) the standard deviation w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F6) the minimum standard deviation of the components’ standard deviations from F5)

F7) the maximum standard deviation of the components’ standard deviations from F5)

F8) the median absolute deviation (i.e. the median of the absolute deviations from the median) w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F9) the skewness (i.e. the asymmetry of a value distribution) w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F10) the kurtosis (expressing the ‘tailedness’ of a value distribution) w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F11) the root mean square (RMS) amplitude w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points, e.g. \( \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \) for a window of size \( n \) containing values of the x component

F12) the covariance of each pair of the three accelerometer components, i.e. \( \text{cov}(x, y) \), \( \text{cov}(x, z) \) and \( \text{cov}(y, z) \) of a considered window of consecutive data points

F13) the minimum covariance of the three covariance values from F12)

F14) the maximum covariance of the three covariance values from F12)

F15) the energy w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F16) the minimum energy of the components’ energies from F15)

F17) the maximum energy of the components’ energies from F15)

F18) the entropy w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

F19) the minimum entropy of the components’ entropies from F18)
the *maximum entropy* of the components' entropies from F18)

- **F21)** the *spectral centroid* (defining the center of mass of a frequency spectrum) w.r.t. a certain component (or the magnitude) of a considered window of consecutive data points

- **F22)** the *minimum spectral centroid* of the components' spectral centroids from F21)

- **F23)** the *maximum spectral centroid* of the components' spectral centroids from F21)

- **F24)** the *spectral energy* (i.e. the sum of the amplitudes of the frequency components after Fourier transformation) w.r.t. a certain acceleration component (or the magnitude) of a considered window of consecutive data points

- **F25)** the *minimum spectral energy* of the components' spectral energies from F24)

- **F26)** the *maximum spectral energy* of the components' spectral energies from F24)

- **F27)** the *spectral entropy* (i.e. the entropy of the frequency components after Fourier transformation) w.r.t. a certain acceleration component (or the magnitude) of a considered window of consecutive data points

- **F28)** the *minimal spectral entropy* of the components' spectral entropies from F27)

- **F29)** the *maximal spectral entropy* of the components' spectral entropies from F27)

- **F30)** the values of the *frequency components* after Fourier transformation w.r.t. a certain acceleration component (or the magnitude) of a considered window of consecutive data points

Almost all of those features are defined on a set of consecutive data points, called *windows*. Hence one data preprocessing step is splitting the overall data stream up into sliding windows as sketched in Figure 3.
Moreover, the features derived from Fourier transforms of the window data require a fixed sample rate. To remedy the aforementioned variance in the sampling frequency provided by the Android sensor API we perform a (linear) interpolation step in the preprocessing phase to get a stable 16 Hz rate. In cases of data gaps or periods of very sparse sensor data we discard these portions and split up the sensor data stream into sub-streams where we could apply the interpolation.

In terms of the features which are based on the window entropy we recognized that the software library we used for the entropy computation\(^1\) returned negative infinity values which suggests that this feature is unsuitable for our data. Due to the fact that some of the classifier implementations we used cannot work on those infinity values we omitted entropy-based features.

Since the sensor readings stem from smartphones which can be oriented in arbitrary directions in space, looking at the individual components did not seem meaningful to us. Hence we focus on features based on the magnitude, or minima and maxima over the components' values of a window.

To get an overview of the impact each feature has we computed the SHAP values [8] expressing how much a feature influences the predicted outcome. A very small SHAP value of a feature here means that this feature has almost no impact on the prediction made by the respective machine learning algorithm and thus could possibly omitted. A diagram with the SHAP values of the 30 most influential features is shown in Figure 4.

\(^1\) [https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html)
Fig. 4: SHAP values of the considered features computed for an XGBoost model

This shows that we covered very important features but it also suggests that the number of features could be further reduced to omit those that contribute only to a small extent. However we kept all the features we mentioned above since this set gave a slightly better performance than the reduced set and we do not face any restrictions on the ML model complexity.

3.3 Smoothing

In terms of the predicted values we observed that there usually are short periods of a trip where the classifiers might predict 'outlier modes', e.g. in cases where a car is not moving due to red light. A made up example could be that for consecutive windows the predictions are

```
car car car bike bus car car car
```

where the correct sequence would be

```
car car car
```
To remedy these outliers a post-processing step was added to smooth the predictions. Here we considered different so called piecewise constant (PWC) denoising techniques [9]² used for image de-noising, as well as more simple techniques like majority vote or median smoothing. To set up a benchmark getting an impression of the effectiveness of each of these methods we considered a made up (but not unlikely) scenario of a commute with several mode changes. The commute comprises:

- a 5 minutes (i.e. 30 windows) walk to train station
- a 30 minutes (i.e. 180 windows) train ride
- a 5 minutes (i.e. 30 windows) walk from the station to a bus stop
- a 15 minutes (i.e. 90 windows) bus ride
- a 2 minutes (i.e. 18 windows) walk to the workplace.

To see how well the respective techniques can cope with noise we randomly picked windows and changed the transportation mode to a false one. The percentage of windows changed was increased from 0% to 100%. After adding noise this way we run the denoising algorithms and compare the restored labels with the labels of the initial scenario. The performance results are shown in Figure 5.

Fig. 5: Comparison of the effectiveness of different smoothing techniques for varying levels of

² The software used for our experiments can be found at http://www.maxlittle.net/software/pwctools.zip
The figure shows that in the majority vote approach with a window size of 20 and 10 iterations (majority_vote_w20_i10) performs well in cases of higher portions of noise. However, it seems shorter periods like e.g. the 2 minutes walk (which amounts to 5.17 % of the overall number of windows) might get erroneously smoothed out. With a reduced window size (e.g. majority_vote_w3_i10) this behavior does not occur, however at certain noise rates the majority votes fall behind majority vote with bigger windows.

In terms of the PWC denoising techniques the jump penalty approach with least-absolute fitting and a regularization parameter of 0.1 (pwc_jump_penalty_least_absolutes_g0.100000) performs best, but still falls behind the majority vote methods. Nonetheless it performs better than the median smoothing. Other PWC denoising techniques seem unsuitable for this scenario. For the task of smoothing the predicted transportation modes for the use case BC2-UC#1 'Modal Split' we evaluated majority vote with a moderate window size, the jump penalty approach with least-absolute fitting and TVD robust. Even though TVD robust only showed a mediocre performance in the evaluation above it provided meaningful results for transportation mode predictions with outliers modes where we did not have true labels but had to rely on human judgements.

3.4 Classifiers
The task of classifying a given accelerometer reading w.r.t. the kind of movement performed was already investigated in the literature (e.g. [2-7]). The work done in this field ranges from distinguishing only very few transportation modes to scenarios like the one we face in use case BC2-UC#1 'Modal Split'. In terms of the utilized sensors the evaluation scenarios range from single to multi sensor settings, sensors mounted on certain spots (e.g. a person’s chest) to the usage of smartphone sensors, as in our case. Usually, good classification results could be achieved with simpler machine learning methods (as opposed to more costly Deep Learning techniques). Thus, to get a first impression we took one of the most widely used machine learning libraries scikit-learn\(^3\) and evaluated the available classification algorithms and picked the five best performing approaches for further evaluation. The algorithms considered in the following are:

- decision tree
- random forest
- support vector machine (SVM)
- AdaBoost
- gradient boosting

3.5 Results
To evaluate the classifiers listed in the previous section we performed a leave-one-user-out cross validation. So, assuming we have a dataset containing labeled accelerometer data of \(n\) users, we take \(n - 1\) for training, try to predict the transportation modes of the \(n\)th and compare the predictions with the \(n\)th user's

\(^3\)https://scikit-learn.org
actual, true labels. We rotate the test user such that we can compute the prediction accuracy for each user, and report the average over all users. The accuracy score is computed as the portion of correctly predicted modes of all predictions made. We present the accuracy of the mere classifier and the accuracies after smoothing where we restrict ourselves to the smoothing methods

- **majority vote** (with 10 iterations and a smoothing window size of 31 predictions), and
- **TVD Robust** (with the regularization parameter $\lambda$ set to 10)

which performed best in our experiments.

The results are presented in Figure 6. Here we show the average leave-one-user-out cross validation accuracy over different window sizes. Additionally we show the average accuracies after smoothing with the majority vote and PVC robust smoothing methods.

![Graph showing accuracy vs window size](image)

**Fig. 6: Average accuracy of leave-one-user-out cross validation for different window sizes and machine learning algorithms**

We conclude that a window size between 60 and 120 seconds would be optimal. Even though considering bigger windows might seem promising accuracy-wise, this would allow only a very coarse grained resolution of the predicted modes where one predicted mode would represent multiple minutes of movement. Hence, shorter walks e.g. between a train station and bus stop could not be captured, but would be either counted as *train* or *bus* movement data, which might blur the classifiers. Overall, we have to admit that there is still room for improvement at this stage w.r.t. the classifiers’ accuracies. This can be attributed to the fact that so far we performed different smaller user studies which means that the amount of training data could still be increased. Moreover, considering the trade-off between the sensor data quality and battery life-time it seems we cannot expect perfect accuracies of the machine
Spatio-temporal Analytics
learning classifiers but also have to rely on the feedback channels explained in Section 5.
4. SPATIAL ANALYTICS

As already stated above for the spatial analytics part we consider spatial background data and citizens’ GPS trajectories to learn descriptive patterns of their commutes w.r.t. the transportation mode used. Hence, the training data here is of a more symbolic nature (as opposed to streams of numbers as in the case of the accelerometer readings) such that we can learn descriptive expressions of spatial relations, such as ‘is near’, ‘is inside’, ‘crosses’ etc. Moreover, making use of taxonomic structures, e.g. by means of ontologies we can even express role and concept hierarchies. Besides this, ontology languages like the Web Ontology Language (OWL)⁴ provide means to express more complex statements like ‘a bus trip is something that starts and ends at a bus stop’, more formally:

\[
\text{BusTrip} \sqsubseteq \forall \text{startsAt.BusStop} \land \forall \text{endsAt.BusStop}
\]

The overall architecture of the symbolic learning part is sketched in Figure 7.

![Fig. 7: Overview of the overall spatial learning architecture](image)

All data sources are eventually processed by the DL-Learner which is a framework for supervised machine learning in RDF/OWL [10]. The citizens' GPS trajectories captured by i-Log serve as training data; the spatial background knowledge stems from the Linked Data mirror of OpenStreetMap, called LinkedGeoData⁵, the RDFized geographical and topological data sets provided by TomTom, and Trento municipality-related open data sources. After linking and fusing the background knowledge datasets (cf. D5.3) the spatial information is mirrored in PostGIS⁶ which is a special purpose database extension of the PostgreSQL⁷ database, designed for spatial data.

A newly developed spatial reasoner component of the DL-Learner makes use of PostGIS to extract spatial relations on the fly, re-formulating them as SQL queries. Thus, there is no need to make such implicit knowledge explicit and store it in the

---

⁴ [https://www.w3.org/TR/owl2-overview/](https://www.w3.org/TR/owl2-overview/)
⁵ [http://linkedgeodata.org](http://linkedgeodata.org)
⁶ [https://postgis.net/](https://postgis.net/)
⁷ [https://www.postgresql.org/](https://www.postgresql.org/)
database. Moreover, this allows some flexibility in terms of the semantics of these implicit relations, e.g. how big the distance between two spatial entities may be to still consider them as ‘near’ to each other. Such concerns can then be captured in a runtime configuration.

All spatial features like streets, points of interest (POIs), areas etc. as well as citizens’ GPS trajectories are modeled as OWL individuals of a certain OWL class, e.g. MajorRoad, ParkBench, NationalPark, or Move. The actual spatial information, like the line string representing the street, the polygon representing an area, the point representing a POI etc. is attached to different OWL individuals of type <http://www.opengis.net/ont/geosparql#Geometry>. Each feature is linked to its geometry through a fixed property, e.g. <http://geovocab.org/geometry#geometry>. A symbolic classifier, e.g. for bike moves, is a learned OWL class expression or OWL ‘concept’ C_Bike which covers all bike moves. ‘Covering’ here means that all bike move individuals b are instances of C_Bike, i.e. for all moves b ∈ Move which are known to be bike moves it holds that b ∈ C_Bike. Ideally, C_Bike is general enough to correctly classify unseen OWL individuals u ∈ Move by performing the instance check for u ∈ C_Bike. The same holds for the learned class expressions for car, bus, train and so on.

The spatial reasoner component described above is used by a dedicated concept learning algorithm to compute the distinctive concepts C_Bike, C_Car, C_Bus etc., as described in the following sections.

### 4.1 Spatial Reasoning

The spatial reasoning is performed in a hybrid manner supporting full OWL reasoning by making use of state-of-the-art OWL reasoner implementations like HermiT\(^8\) or Pellet\(^9\), adding a set of ‘virtual’ OWL object properties expressing spatial relations which are not explicitly stored in the knowledge bases. Besides the set of relations defined in the Region Connection Calculus [11] further relations were added. The full list can be found online\(^10\). The semantics of many of these relations are fuzzy. To give an example, the statement whether two spatial things are near to each other might depend on a person’s interpretation or use case requirements. Accordingly, we made such spatial relations configurable, e.g. by allowing to state that other spatial things must be within the radius of 10 meters to count as near. Since the OWL object properties expressing these spatial relations are virtual and computed at runtime, adjusting a relation’s concrete semantics, e.g. by changing the near-radius to 5 meters, does not require any changes of the background data.

To give an example of spatial reasoning in the light of the use case BC2-UC#1 ‘Modal Split’, we may consider the relation runs along which states that one line feature runs along another line feature. This relation is different from the equality relation for line features since it allows having differing lengths, start and stop points

---

\(^8\) [http://www.hermiT-reasoner.com/](http://www.hermiT-reasoner.com/)

\(^9\) [https://github.com/stardog-union/pellet](https://github.com/stardog-union/pellet)

Spatio-temporal Analytics

and so on. So it rather requires that some sub-segments of both line features are ‘similar’. For the considered use case this relation can be used to capture the road types a user went along to infer, e.g. that it must have been a train ride if the movement ran along a feature of type *RailroadTrack*. But this comes with some further requirements since the GPS trajectories are usually noisy. Hence the implementation for the *runs along* relation has to cope with certain degrees of fuzziness. Figure 8 shows a segment of a bike ride performed by one of the authors, captured during a QROWD user study.

![Fig. 8: Bike ride trajectory exemplifying the spatial relation 'runs along' on the results resources resolves to <http://linkedgeodata.org/triplify/>](image)

The actual movement trajectory is given by the red line. The street segments returned by the spatial reasoner component, if asked for spatial individuals which the bike ride ran along, are colored and highlighted\(^\text{11}\). Here, a tolerance in meters can be set to adapt to accuracy of the captured GPS trajectory.

### 4.2 Spatial Concept Learning

The learning part to find a distinctive OWL concept covering, e.g. all bike trips, is performed by algorithms developed for the DL-Learner framework for supervised machine learning in RDF/OWL. The main procedure we follow here is to start with a general concept, which might cover not just movements performed by the transportation mode we want to learn a concept expression for, but also further modes, and try to refine it so that it only covers and thus serves as classifier for e.g. bike trips. *Refinement* here means that with each step we want to get a more special OWL concept which better describes the respective mode to learn the classifier for. A (made up) sequence of refinement steps for a bike trip could look like this:

Spatio-temporal Analytics

SpatialFeature

- LineFeature

- LineFeature $\sqcap \exists \text{runsAlong,}$ $\top$

- LineFeature $\sqcap \exists \text{runsAlong.Road}$

- LineFeature $\sqcap \exists \text{runsAlong.Cycleway}$

- ... given that Cycleway $\sqsubseteq$ Road

To handle virtual OWL properties like runsAlong we extended the existing learning algorithm CELOE [12] as well as its refinement operators performing the refinement steps as exemplified above.
5. CROWD FEEDBACK

So far we covered the machine learning part of the spatio-temporal analytics tasks for the use case BC2-UC#1 'Modal Split'. However, due to the constraints mentioned above and during the bootstrapping phase there might be cases where we need an efficient and effective channel to a ‘human-in-the-loop’ to improve the analytics results or guide the analytics component. In the exemplified use case this occurs whenever the mode detection can only vaguely determine a transportation mode, e.g. because movement or accelerometer patterns were not distinctive enough for the trained classifiers, or in cases where the input patterns were never observed before. The most accurate human guidance one can get in this case is the direct feedback from the user who generated the sensor data through the i-Log mobile app. i-Log allows to get in touch with an app user through questions or extended dialogues. Whereas questions allow to choose predefined answers, extended dialogues allow e.g. to define points where a user changed the transportation mode. Figure 9 shows an example for each of these two categories.

On the left-hand side the i-Log app presents a map and textual information about the trip we detected. The question a user is asked serves as confirmation that our preprocessing, i.e. detecting and extracting the commute parts of a full day sensor reading, was correct. In a similar manner we asked users to select which of the transportation modes were used for a detected trip such that they had to click on either car, motorcycle, train, bus, cableway, bicycle or walk. The second image in Figure 9 sketches an extended dialogue to pinpoint at which position on a GPS trajectory the mode changed, and to which transportation mode.

In our current user study design we process the citizen sensing data on a daily basis. The i-Log app collects sensor data during the day and uploads it whenever a wifi
connection is established. Accordingly, for the majority of cases we can assume that data was uploaded in the evening, after a citizen came home from work. A fallback point for those citizens who have not returned in the evening would be the early morning of the next day. Hence, we can run our analytics process during the night to cover most of the citizens, and maybe a second time in the early morning to process the data of as many citizens as possible. In the usual case the gathered modal split information is stored in the QROWD DB. In cases where citizen feedback is needed we make use of the i-Log feedback channel aiming at getting guidance or missing labels to improve the machine learning algorithms by asking the citizens about their yesterday’s trips.
6. CONCLUSIONS

In this deliverable we touched on details regarding the spatial and temporal analytics capabilities of the QROWD platform. We showcased the analytics techniques applied in the use case BC2-UC#1 ‘Modal Split’ and introduced the data sources as well as how data is preprocessed. We showed how we train classifiers with accelerometer and GPS data and discussed different design decision in the overall workflow. Furthermore we gave a brief overview of options for user feedback to guide and improve the analytics components.
REFERENCES