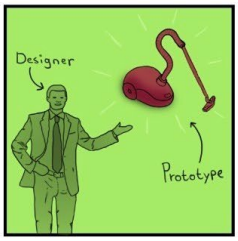
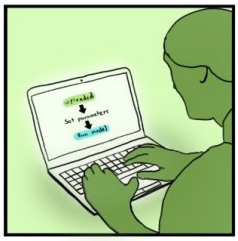
Machine Learning for Designers - Report

TI3150TU - Group 7



Designer wants to test new prototype

Sensor placed on hand of user gets movement data during user testing



Designer uploads received data to our notebook and tunes required parameters before running classification algorithm

Algorithm classifies data based on user actions. Using anomaly detection, the algorithm captures strange user behavior which the designer can then analyze further

Skip Doorn - 5356202
Alan Roukema - 5211883
Ian Tiemann - 5241200
Sophie Vlot - 5329922
Gijs Volkers - 5295920
Timo Zunderman - 5293707

TU Delft

Index

1. Introduction

This document is a means to follow the developing process from start to finish. All information about the project that is not included in the actual product (Notebook and website) can be found here. For weekly planning, meeting notes, and completed items, please refer to the Notion workspace.

1.1 Background and Project Description

Data-driven design offers new ways of using data to create products, services and experiences that are better at meeting users' needs. Technology is currently in a state where using cheap sensors to generate information about the product, and its usage can already significantly impact the user experience of such products in a positive way (Jagtap & Duong, 2019). Data collected by sensors from prototypes or even finished products are a treasure trove of information. However, sensors generate many data, and when looking at multiple sensors across multiple products, it can be hard to get any helpful information. Even if the Designer can deduct surface-level patterns, the deeper levels will always be hidden from a person analysing at this scale. Machine learning techniques can help create insights that are not visible by looking at the raw data. Moreover, machine learning can make this data insightful by visualising an overview. The information gathered from this can be used to develop the product further, adapting it to how it is utilised in real life.

With the knowledge of basic machine learning concepts, we are asked to work on a project based on these insights. This project is led by the clients/supervisors Jacky Bourgeois, Dave Murray-Rust and Kostas Tsiakas, with the support of Teaching Assistant Matej Havelka.

The use of data to improve products, services and experiences is not a new concept in the design world. Nowadays, a couple of high-quality sensors are relatively cheap and easy to purchase. However, hidden in raw data is valuable information that is hard to extract by hand. The goal is to make this more accessible and less time-consuming for a designer without in-depth knowledge of these techniques with the use of AI. This project uses a combination of AI techniques like active learning, dimensionality reduction and novelty detection to analyse the data and generate useful information for the Designer in a fraction of the time it would take by traditional means. The Designer gets the relevant types of information - statistics about how the product is being used and unexpected behaviour.

2. Development Process

In this chapter, a complete overview of all the technicalities behind the code, as well as design choices, can be found.

2.1 Data Collection

To collect data, sensors were used, while activities were executed in different contexts. These generated datasets can be used to write and test the written code. The data collection process will be described here, as well as the different contexts in which the data was collected.

The first context is walking. The activities included walking up/down the stairs, running, and walking normally. In the beginning, the data was collected via mobile phones since all modern mobile phones have an accelerometer and a gyroscope, and a user could also use this. It wanted to collect real-life data, so it was collected in the centre of Delft, where the data was collected by walking through the city centre and climbing the Nieuwe Kerk. A total of five people climbed the Nieuwe Kerk, one person recorded data while walking through the city centre, and one person recorded data while running. Recording data with multiple persons results in a more robust model since everyone walks in a slightly different way. During collecting the data, it was made sure that everyone used the same app to collect the data and that the phone was held in the same position and was titled in the same way by everyone. If this was not done correctly, the model could not recognise that the same action was executed because the values of all the sensors would be significantly different. After this first data collection session, it became clear that the app only recorded about twenty-second clips. This app flaw resulted in the decision to use a GoPro from now on to record the data. Using a GoPro has two advantages: there is no limitation on the recording length, as long as the video file fits on the sd-card inside the GoPro and a video is made simultaneously. As explained in a later part of this report, the video will be used by the user to train the model. The way the GoPro was used and how the data was extracted from the video file can be found later in this chapter.

The second context in which data was collected was vacuum cleaning. Again the sensor placement was discussed, and it was made sure that the GoPro was placed in the same way each time. In total, three different people vacuum-cleaned their room or house while recording a video with the GoPro. At this point, it was possible to extract the accelerometer and gyroscope data from the video file. During the last stage of testing the product, it was decided to record two new datasets. It was wanted to test if the model could also work in yet another context, namely product-based activities. The data was collected while holding a mug, which the actions consisted of being still, walking while holding the mug, picking it up, putting it down, and drinking from it. After testing this data, it was concluded that the model also could be used for prototypes where there are instances of no activity, as long as they are marked as an activity. The second dataset that was collected in this last stage was cycling. This was done to determine the different activities on various textures of the ground. The different activities revolving around cycling were standing still, staying at a constant speed, accelerating and slowing down. The different types of ground consisted of asphalt,

clinker bricks and concrete clinkers. Figure 2.1 shows the respective types of roads. All datasets are submitted along with this document.

Figure 2.1: Asphalt, clinker bricks, concrete clinkers

2.2 Importing GoPro Files

To make the sensor data ready for preprocessing in the Notebook, there are some steps to be taken. They will be explained in this chapter.

2.2.1 Extract the data from the MP4

A GitHub repository was found that could extract the GoPro telemetry data from the video file into a .json file (JuanIrache, 2022). Another source explained how it was possible to use the extraction tool to get the JSON file (G., 2022) containing the telemetry data. The code can be found in the repository under GoPro in the file '*GoPro_Telemetry_Code*". In this code, the GoPro video can be given as input, after which it saves the JSON file. If the explanation of G. is followed, it is possible to get the same code as was used. The code is in Appendix M, but it might be necessary to download the necessary packages as explained by G. Unfortunately, there was not enough time to explain how the installation should be executed extensively. We found this was a little outside the scope of the goal of this project. It is expected that designers probably will find this a little too hard to use, but with some work, this process could be improved in the next iteration of this project.

2.2.2 Transform the data into usable data

The JSON file generated in the extraction step was not usable for the algorithm since the preprocessing expects a certain format. In order for the program to work, it is necessary to transform the data streams ("GYRO" and "ACCL") into two separate CSV files, including the timestamps at which the data was sampled. A small Python script called "*jsontocsv.py*" can be found under the GoPro tab in the repository, that does exactly this. It reads the JSON file and extracts the data streams that are needed into separate files. First, the file path of the JSON file needs to be entered together with the sensors that are wanted to be written to the CSV file. The script then saves the desired datastream as a CSV file that can be used by the next part of the algorithm, the preprocessing part.

{"1":{"streams":{"ACCL":{"samples":[{"value":[9.437799043062201,0.17942583732057416,3.3636363636363638],"cts":0,
"date":"2023-01-16T10:53:45.000Z","sticky":{"temperature [°C]":25}},{"value":[9.490430622009569,0.18181818181818182,3.
351674641148325],"cts":5.128205128205129,"date":"2023-01-16T10:53:45.005Z"},{"value":[9.421052631578947,0.
20574162679425836,3.3157894736842106],"cts":10.256410256410257,"date":"2023-01-16T10:53:45.010Z"},{"value":[9.

Figure 2.2: extracted JSON File.

```
1  0.0,9.437799043062201,0.17942583732057416,3.3636363636363638
2  0.005,9.490430622009569,0.18181818181818182,3.351674641148325
3  0.01,9.421052631578947,0.20574162679425836,3.3157894736842106
4  0.015,9.411483253588516,0.19138755980861244,3.327751196172249
5  0.02,9.411483253588516,0.1937799043062201,3.3660287081339715
6  0.025,9.385167464114833,0.23444976076555024,3.382775119617225
```

Figure 2.3: "ACCL" (accelerometer) datastream in a CSV file.

2.3 Preprocessing

After the data collection, the data needs to be preprocessed in such a way that it is usable for the machine learning model. The raw data that was collected is not suitable for these models because it is both not in the right format, and there has not been any feature extraction.

The format is important because Python needs to be able to work with the data in a way that is always similar, regardless of the original. But more important is the feature extraction. This is a process in which values are derived from the raw data. An example could be taking the average from a set of numbers. This will make sure the model has enough relevant data, which improves accuracy and reliability, reduces overfitting and speeds up training. The process of preprocessing that is used during this project is described below.

2.3.1 Windowing

First, the data is windowed, i.e., taking a small part of the data from which the features are extracted. To have as many windows as possible, sliding windows were used, which means that the different windows overlap to make sure activities are caught, even if they don't fit one of the windows. As can be seen in Figure 2.4, window W2 overlaps partially with W1 and W3. After processing a window, the window is shifted some amount of time in the future, and the features are extracted until the end of the full data stream is processed. In the figure, the window size is two seconds and the slide time, or offset, is one second. By using a sliding window, each window will be classified with regard to what is being done in that particular window.

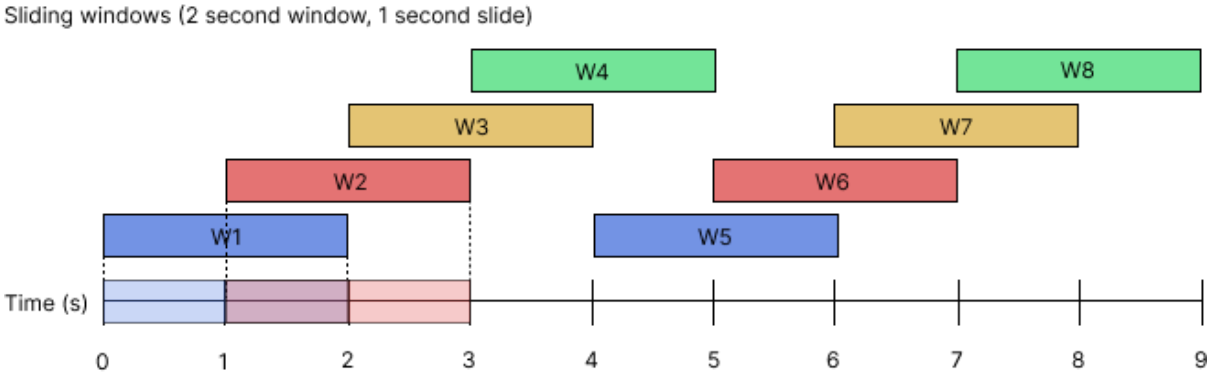


Figure 2.4: Visual representation of sliding windows.

For example, when an activity is happening between one and three seconds, windows W1 and W3 would not be able to correctly classify this activity with high certainty. W2, however, can because it sees the whole action.

2.3.2 Time domain

After a window is made and the data that is in that particular window is collected from the files containing the sensor data, the features are extracted. In total, five features from the time domain and three features from the frequency domain are extracted. The time domain is the graph that can be obtained when the sensor data is plotted. It visualises the amplitude (y-axis) set out against the time (x-axis). An example of a graph is shown in Figure 2.5, where a phone moved while recording the accelerometer data.

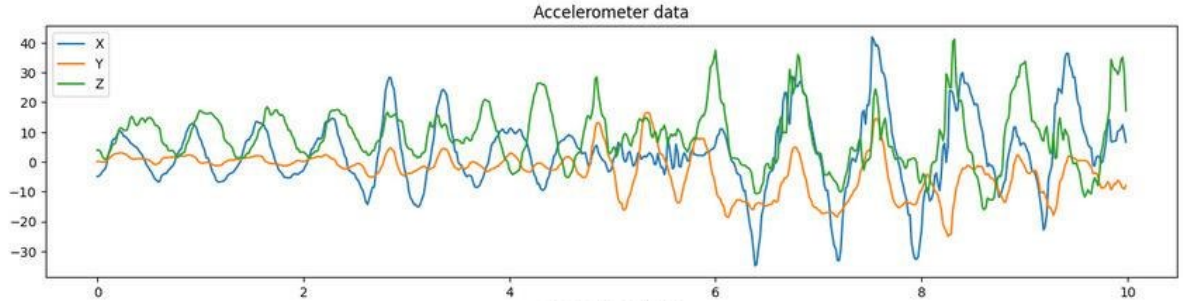


Figure 2.5: Graph of accelerometer data in the time domain.

The features per axis of each sensor that are extracted in the time domain are represented in Table 2.1.

Table 2.1: Features in the time domain and their explanation.

Feature	*Explanation*
Minimum	The lowest value in the window
Maximum	The highest value in the window
Average	Average of the values in the window
Standard deviation	Standard deviation of the values in the window
Area under the curve	Area that is between the curve and the x-axis

2.3.3 Frequency domain

The frequency domain is more complicated. Each signal can be expressed as a sum of sines and cosines with different frequency components. These individual frequency components and their amplitude can be found by the use of a Fourier transform. The process of calculating the components will not be explained in this report, but it is a well-known method that is used to extract wifi signals, for isolating audio and improve the quality of images for example. In this preprocessing step, it is used to extract additional features that are hidden in the time domain.

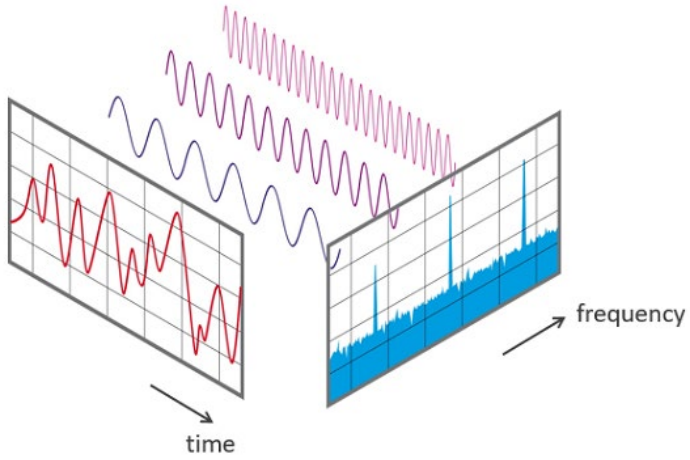


Figure 2.6: Visual representation of time and frequency domain and their relation to each other (Fourier transformer)

After the Fourier transformation, the contribution of the different frequencies that are part of the time domain can be seen. This data can give the most important frequency, which could be different for some or every activity. Figure 2.7 is an example that was extracted from data collected for this project. The extracted features can be found in Table 2.

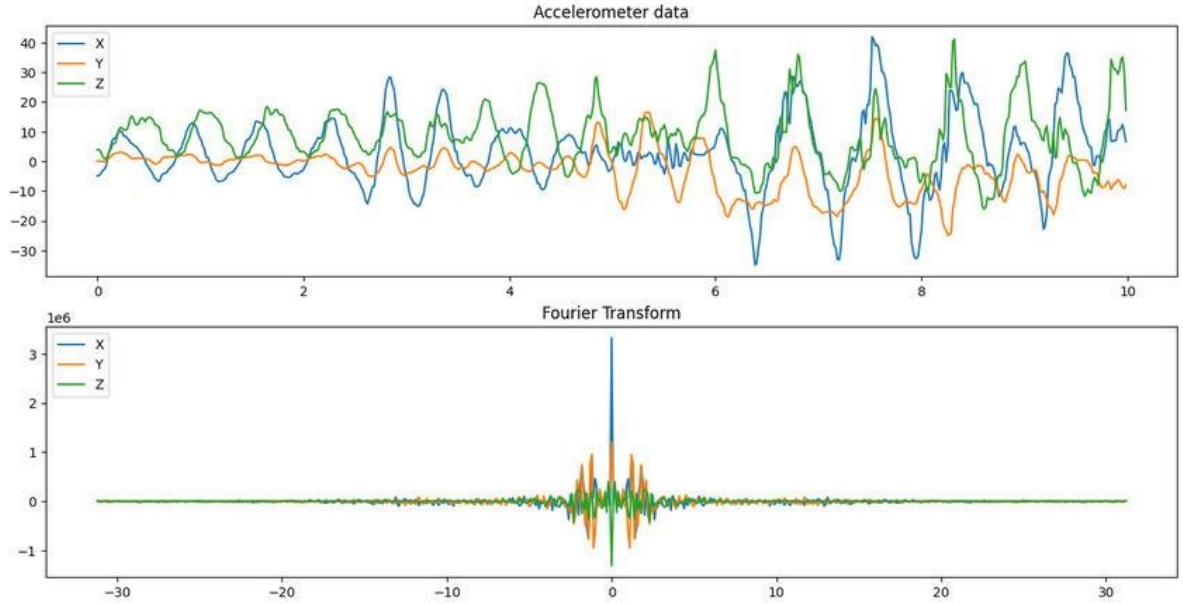


Figure 2.7: Graph of accelerometer data in the time domain and frequency domain (Fourier transformer)

Table 2.2: Features in the frequency domain and their explanation

Feature	*Explanation*
Centroid frequency	The frequency where the energy of the frequencies to the left is equal to the energy of the frequencies to the right, excluding 0Hz
Maximum energy	The maximum energy, excluding 0Hz
Peak value frequency	The frequency with the highest amplitude, excluding 0Hz

After having extracted all these features from the X, Y and Z axis of both the accelerometer and gyroscope, there are 48 extracted features (8 basic features * 3 axes * 2 sensors).

After extracting features from every window in the time and frequency domain, preprocessing is done, and model training can start. In the design process, the appropriate machine learning model needed to be selected first.

2.4 Model Selection

A model needed to be selected which the product was going to use to classify the sensor data. All the models come from the scikit-learn package. To come to a decision, a selection of models was trained and evaluated on previously collected data. The selection of these models was based on research papers that showcased which models are useful when working with accelerometer data. From different papers, the random forest seemed to be the most promising (Nurwulan & Selamaj, 2020) (Lavanya Devi & Vizianananda Row, 2017). The tables below show the models that were used and their respective accuracies, together with the time it took to train the model. The models were tested on two different already preprocessed datasets, the walking dataset and the dataset from the mug. The results of the first dataset are shown in Table 2.3 and the second in Table 2.4.

Because k-means learns in an unsupervised way, it does not have a train or test accuracy. The test accuracy is calculated by using a test set and by hand comparing every possible combination of clusters and picking the best-case scenario. So for 3 categories and clustering unsupervised, there are (3 * 2 * 1 =) 6 possible combinations.

Table 2.3: Accuracies for dataset 1 - Walking

Model	*Train accuracy*	*Test accuracy*	*Time (s)*
K-neighbors	0.89	0.80	0.12
SVC	0.85	0.90	0.25
SGD	0.72	0.74	0.01
Decision tree	1.0	0.97	0.01
Bagged classifier (using 100 decision trees)	1.0	0.98	0.76
Random Forest	**1.0**	**0.98**	**0.24**
K-means	N/A	0.81	0.10

Table 2.4: Accuracies for dataset 2 - Using a mug

Model	*Train accuracy*	*Test accuracy*	*Time (s)*
K-neighbors	0.98	0.94	0.15
SVC	0.97	0.93	0.26
SGD	0.96	0.91	0.02
Decision tree	1.0	0.93	6.46

Bagged classifier (using 100 decision trees)	0.97	0.93	4.59
Random Forest	**1.0**	**0.96**	**0.47**
K-means	N/A	0.45	0.21

From these tables, it can be concluded that the random forest model is the best option for both use cases. The first reason for this is the high accuracy on the datasets, namely 1.0 on the training set for both the datasets and 0.98 and 0.95 on the test sets. The second reason is the time it takes for the model to run. Even though the bagged classifier has about the same accuracy as the random forest, it takes significantly longer for it to be run compared to the random forest model. As active learning will be used (which is explained later on), the entire model will need to be re-trained multiple times. If training takes a lot of time, this process will take too long to be viable.

The decision tree classifier also performed quite well on the trained and tested data, and the time it took was also very fast. However, this model wasn't used because it could potentially perform worse in data that is not easily separable by linear decision boundaries. The random forest model will create a more smooth transition between the two classifications. If data is provided that is more intertwined, the random forest can potentially separate these activities better. The results will also be more stable due to the use of many decision trees (Talari, 2022).

2.5 Active Learning

Active learning was used to train models. This is useful for decreasing completion time because the Designer will only be asked to label a minimal amount of data points. Finding a way to decrease time spent on labelling data is extremely useful in this case because recording just minutes of sensor data already generates thousands of data points.
The first stage consists of selecting starting points and labelling them. After that, an algorithm is put in place to select just the least certain unlabelled points, which the Designer will then have to label. This is the second stage. This way, the dataset just has to be corrected when it is not sure, and all other data points are automatically labelled. This can save a lot of time for the Designer, labelling every point by hand the traditional way. These self-labelled points are added to the training data, and the model re-trains on this data once for every iteration.

To select the starting points, a combination of functions was used. One function randomly selects starting points, and another uses an algorithm to find certain data points (Appendix E). The algorithm that was used is k-means clustering. If a point lies exactly on its cluster centre, the certainty that it belongs to that specific cluster is the highest. The further it lies from the centre, the more chance that it could actually belong to another cluster (and k-means made a wrong prediction). To minimise iterations, the active learning pipeline should start training on the most certain points. This information was used to detect them and combined with the other function. Each activity got selected to be a part of the initial training data in the next part, where a model is trained on the pool of labelled training data.

After the first iteration, a function is used that computes the least certain unlabeled sample from the training pool. This function can be found in figure E.1, and it works by computing a margin. The margin is the difference between the probability of the predicted class and the highest probability of another class being the correct prediction. If there's a list that looks like this: [0.50, 0.49, …], it means that the certainty of the sample being the predicted class is 50%, and the certainty of the label being another class is 49%. This needs to be avoided as much as possible, so this list gets computed for every unlabelled sample, and the sample with the smallest margin gets added to the training data. The Designer is asked to label this point, and in turn, it gets added to the labelled training pool, and the model re-trains again. As a result, the model will make increasingly better predictions for unlabeled points. In Appendix D, there are graphs that show how metrics like test accuracy and the Gini index improve increasingly.

As for the approach for testing and checking for overfitting, a combination of different evaluation metrics was used. As random forest classification was used, the Gini index is a measure of the purity of splits. As seen in Appendix D, the Gini index approaches zero more after each iteration. This means that splits get purer, but overfitting has to be avoided. This is why the decision was made to label the entire dataset and set aside some test data to compute test accuracy. The test accuracy shows that there's no case of overfitting. Lastly, the ambiguity was computed using the margin. In the actual iterations, it gets computed for every sample, and the samples with the smallest margin were added to the training data. The margin included in the graph is the lowest margin found over all samples. This is because the aim is to have a list that looks like this: [1.0, 0.0, 0.0, …], the lowest margin is a kind of 'worst case', and the lowest margin is ideally as high as possible.

In order to get a better idea of the accuracy of the active learning model, data was collected for a second use case: a GoPro was tied to a mug, as explained before. Five labels were chosen, namely: the mug is staying in the same place, the mug is being picked up, the mug is being put down, somebody is drinking from the mug, or somebody is walking with the mug in one of his hands. In total, 3 minutes and 11 seconds were recorded, resulting in 82 actions, where each action was one of the five predefined labels. Labelling these 82 actions by hand took approximately an hour while labelling about 125 samples of 0.7 seconds only took about 5-10 minutes. It is important to note, though, that during the user testing, it became clear that designers take a little bit longer to label these samples because they had never done it before. Even though it might take longer for an inexperienced user, it still took no more than 20 minutes, much less than the hour it took to label all the data by hand.

Logically, the decrease in time also results in a decrease in accuracy. Letting the model predict each window resulted in an error rate of 102/1058 or 9.641%. 52.57% of the time, the mug was standing still on the table. This was our dummy classifier (always predict 'still', with an accuracy of 52.57%) and thus the score to beat. Analysing the wrongly labelled samples, it became clear that the samples that are wrongly classified mostly are the labels where there is a transition from one first label to another label. Those samples are often difficult to label (for labelling them via active learning as well as labelling them by hand) since those samples are regularly a combination of two actions. This means that if the actions take longer, the accuracy will go up.

In conclusion, even though the model made from active learning was less accurate than labelling the data by hand, it was accurate enough to get an idea of what happened during the recording. This can be seen in Figure 2.8. Each line corresponds to a window, with the colour representing one of the labels. In the figure, it can be seen that most times, the mug first stands still, then gets picked up, somebody drinks from the mug or walks with the mug, the mug is put down, and it stands still again.

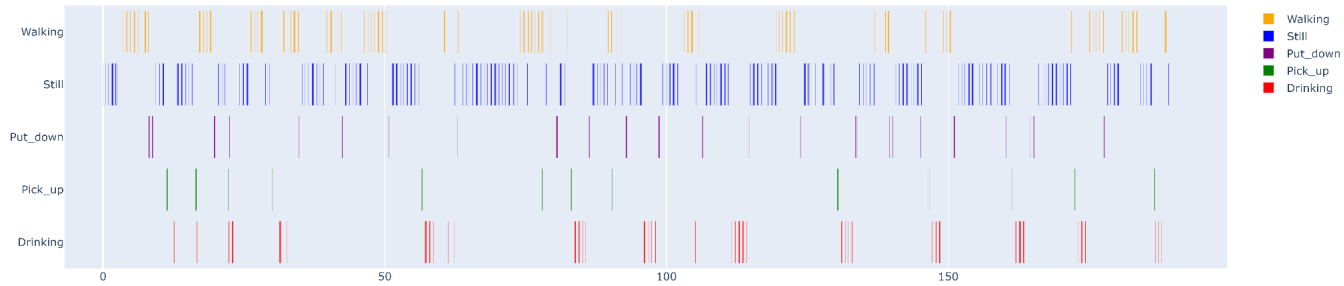


Figure 2.8: Time distribution graph

In order to improve the accuracy, multiple measures could be taken. The easiest solution would be that more samples are labelled since this would only require re-training the model. Other possible more rigorous measurements could be, choosing different, more distinct labels or recording more data.

2.6 Novelty Detection

LocalOutlierFactor from *Scikit-learn* was used to investigate the possibility of novelties after training and to evaluate the models' performance. The novelty detection displays the points that are the most different from the rest of the data. 10% of the data that is the most different is displayed by default. This data is displayed by showing the video of that data point to the Designer. With these videos, the Designer can see if there is any unexpected behaviour, which they can analyse by watching the videos. The file name and timestamp will be printed so the Designer can take a look at the video themselves. The code used for novelty detection can be found in Appendix H.

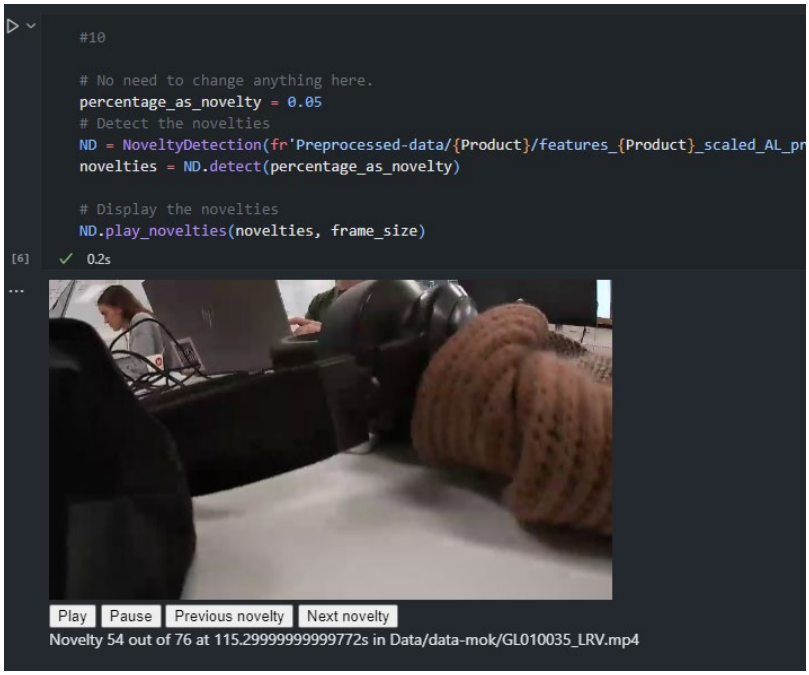


Figure 2.9: An example of the novelty detection media player

2.7 PCA Visualization

PCA was used as a dimensionality reduction method to compress 48 dimensions into 2. This is to visualise the dataset in a plot. The dataset is plotted during active learning to visualise the point being classified and to see if any clusters are forming with regard to the different labels. This will give the Designer some information about the dataset in general, what they have already classified and the current point. However, the PCA cannot be used to draw any conclusions because the axes don't have meaning. Consult Appendix I for the code for PCA and the plotting.

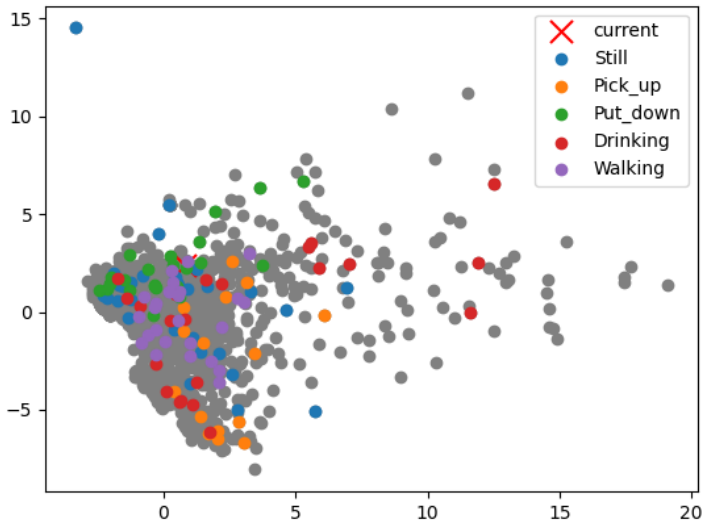


Figure 2.10: PCA analysis graph showing the points that are labeled

Figure 2.10 shows the points that are labelled and the current point that is waiting to be labelled. This graph is shown in the Notebook after about 25 points (depending on the settings and the number of labels) are labelled by the user and are updated in real-time during active learning. This way, the user can get insights into potential clusters.

2.8 Notebook and Instructions

The choice was made to make a Jupyter Notebook (from now on abbreviated to Notebook) that the Designer can use to execute the written Python functions. A Notebook is relatively easy to use since text and images can easily be added around blocks of code that can be executed. In this way, it can be briefly explained what the user should do without having to go back to another document for instructions. Another option was to make a graphical user interface in Python. This would require much more time since it was needed to learn how to do this. For the instructions, the decision was made to keep the text in the Notebook as minimal as possible. Just the essential step-by-step instructions were put in the Notebook. Other important instructions, like a word list and introduction, would be found elsewhere. This way, the Notebook is as uncomplicated as possible. The instructions that are specific to one code block are included in the Notebook. After testing this workflow, the most efficient distribution of instructions was found, where the Designer would have to switch the least possible amount between the Notebook and other instructions.

This decision was made because complete instructions combined with code blocks would simply be too overwhelming for designers without any coding experience. Scrolling back to consult previous instructions is also limited this way. After having written all the instructions, the information was kind of scattered across files. All the instructions that were not included in the guide had to be put in one centralised location. At the advice of the client, GitHub pages were used to create a website. This way, the Designer can keep this website open in a separate tab and can easily consult it at any time they feel the need. Pages were also linked inside the text to make for easier navigation through the website.

2.9 Other Design Choices

To ensure that the Designer understands what steps will need to be taken in the Notebook, a flowchart showcasing all the various steps is added at the beginning (Figure 2.11). Adding to this, an arrow shows where the Designer is currently in the process. This way, the users can get a better overview of the Notebook. These design choices were made after multiple surveys suggested that some more visual cues instead of just text could make everything a bit more clear.

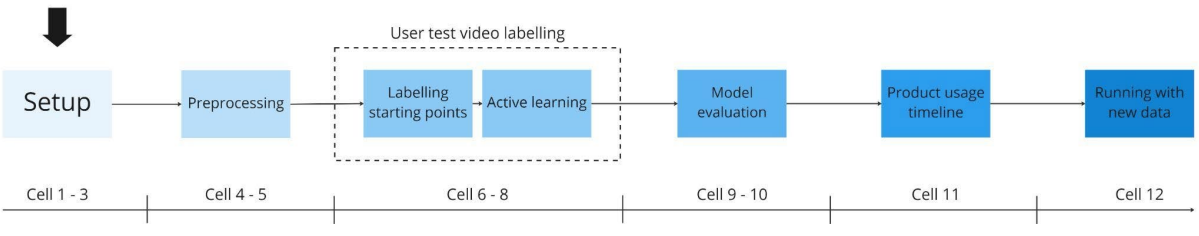


Figure 2.11: Flowchart of steps in Notebook. The current step is 'setup.'

At the beginning of the project a brainstorm was conducted to decide on what type of visuals would be interesting for the designer, one of which was a bar chart that shows the amount of incorrectly and correctly labelled datapoints per label. However, the code was written before there was decided to use active learning, rendering the bar chart useless as there are no correct labels to compare the predicted labels with. It could have been implemented in active learning but time constraints caused it not to be a priority. This visual, however, can be a useful for a designer. In further improvement on this project this chart may be a useful addition. The bar chart visual showing predictions of collected data can be viewed below. As can be seen all data is correctly labelled.

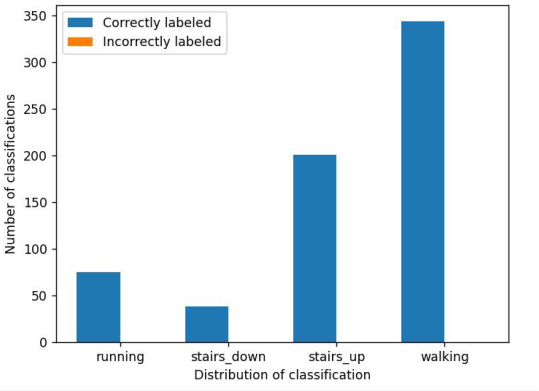


Figure 2.12: Bar chart of label predictions

3. Evaluation

In this chapter, the evaluation of our product through user testing and requirements validation will be described.

3.1 User Testing

The evaluation should be designed in such a way that test subjects are able to complete it in a reasonable amount of time, which is why a given dataset was presented instead of having the focus group collect data during the evaluation. Deciding to have test subjects answer the questions in a setting like this will help actually make observations as well. Watching actual designers use the product will give more useful information, as opposed to just analysing the survey answers after completion. Another advantage of this approach is that expectations can be set and evaluated in a systematic way by defining them beforehand. This gives another measure of how the product is received by designers. For example, the duration of the evaluation (completion time) can be compared to how long it should reasonably take, assuming the product provides comprehensible instructions.

The user evaluation was carried out in an iterative way. Two surveys were taken off very extensively, one-on-one. All observations were written down, and the instructions were adjusted accordingly after this first stage. The second evaluation happened after the guide was improved based on the feedback from the first evaluation. 9 designers tested the product while notes were taken about their process. To document the observations, some preliminary expectations and metrics were created, and any errors or questions during the process were written down. As for the scale, it was decided to stay consistent with a 1-5 Likert scale on each question (Kaptein, 2010). Because questions about experience are very subjective, it's important to create a way to get the least ambiguous answers while still getting nuanced replies. A 5-point scale was chosen for user-friendliness and because it includes a midpoint for neutral attitudes. In this case, a 5-point scale is more user-friendly because it avoids designers getting stuck on a question because they can't decide on an answer. This would have a significant influence on reliability because this evaluation will already take quite long to complete. Getting a wide variety of numeric results is also less important, as notes were taken during the evaluation as well.

The full survey results can be found in Appendix C. Table 3.1 shows some metrics about the numeric questions from the evaluation survey. The cells that are highlighted grey show the results of questions about ability and difficulty (where 5 is the easiest, and they are well able to execute the tasks), while the unmarked cells represent questions about the subjects' understanding (where 1 represents no understanding at all). The latter is included to investigate if there's any correlation between understanding and how difficult a subject found a step. From these results, it becomes clear that this is not the case. This means that it can be stated that the instructions make for better usability, regardless of experience. Question 4.2 is about the general difficulty of using the product, and the average score is equal to 4.27. The numeric survey results are positive, even for designers that don't understand machine learning at all.

Table 3.1: Survey results - numeric questions

	Mean score	Minimum	Maximum	Modal value
Loading data				
1.2	4.17	2.5	5	5
Preprocessing				
2.2	4.27	2.5	5	5
2.3	2.61	1	5	2
Getting output				
3.3	4.56	3	5	5
3.4	2.61	1	5	2
Final thoughts				
4.2	**4.27**	**3**	**5**	**5**
4.3	3.05	2	5	2

The expected completion time was 30 minutes, while most evaluations took around an hour. The most occurring feedback was that the main goal of the program and the steps were not really clear. Also, it was unclear how you could edit paths in the file. Moreover, there was some misunderstanding about if something had run and how long cells would take to complete. For instance, it was unclear how many videos they had to label and how far they were in the process. The general expectations were very low, and most designers had no idea what the process would look like before reading through the website. Some designers thought they had to code something themselves. After completion, subjects found using the application surprisingly easy. The general consensus was that it was easy to follow the steps and that they didn't need to know anything about the inner workings. There were some modifications made after the feedback. A background page was added to the website to make for a better understanding of the main goal. Another functionality was added to the Notebook to display the samples that still had to be labelled. The code blocks were improved on readability. Lastly, the instructions were improved based on all the questions raised by the test subjects. The biggest risk of this product is the user not being able to use it, and this is minimised as much as possible by doing this. The risk of the user not reading the text well is not something that is preventable.

3.2 Requirements Validation

The table below is an overview of how the requirements were validated and what changes were made to the validation approaches. The requirements can be found in Appendix F.

Table 3.2*:* Requirement evaluation table

Requirement	*Evaluation*	*✓/X*
M.1	Implemented gyroscope data in week 7 due to setting priorities. No unforeseen circumstances occurred after implementing this functionality, so	✓

	no further changes	
M.2	No changes, finished data collection and processing sooner than expected	✓
M.3	No changes, finished before the agreed deadline. The model could already distinguish 3-4 activities by week 6, namely walking, running, walking up the stairs (and walking down the stairs)	✓
M.4	No changes, validation can be found under *Testing models*	✓
M.5	No changes, justification under *Model selection*	✓
M.6	No changes, our dummy classifier accuracy for the mok was 52% and our model accuracy was 91%, so our model outperformed the dummy classifier	✓
M.7	No changes, design choices can be found under *Notebook and Instructions*	✓
S.1	No changes, annotations are present	✓
S.2	No changes, explanation under *PCA Visualization*	✓
S.3	By week 7, we had realised that linking a report grade to user experience might not make as much sense as we had initially thought. After discussing this with the client, we decided to change the metric for S.3 to observations, combined with a (statistical) overview of the most informative results (Appendix C)	✓
S.4	We tested one model architecture, and it worked for the initial two contexts (*Testing models)*. To expand upon this, we collected more data from a different context (*Data Collection)* and tested it	✓
C.1	We haven't implemented anything to accommodate uploading or downloading data from the IoT-cloud. This was beyond the scope of our project.	X
C.2	Because of time constraints but also due to some lacking knowledge we couldn't implement self-learning techniques.	X
C.3	No changes, explanation under *Novelty Detection*	✓

4. Final Thoughts

4.1 Reflection

4.1.1 Summary of logbook

In Appendix L, there is detailed documentation of our weekly progress - a summary will follow in this section. By week 4, we had already finished the preprocessing steps of the process. This involved the steps needed to upload and preprocess the accelerometer data gathered during data collection, which all happened in the same week, week 3. Since we had already finished building and testing our active learning model, we'd finished M.3 ahead of schedule. At this point, our model was already able to distinguish between at least 3 different given activities based on the data we collected in week 3. We decided our model would show the timestamp of an activity, and before Christmas break, we could present the Designer a GIF of the activity at that timestamp. At this point, our model was consistently able to get close to 95-100% accuracy on the training, as well as test data we gathered ourselves.

4.1.2 Teamwork

In this section, we will summarise how the development of our product has progressed over the past weeks. The first four weeks ran smoothly; we were consistently ahead of schedule. Aside from having to finish preprocessing instead of starting to select models first, we were able to stick to our planning and even did some extra work. Unfortunately, every team member got the flu one by one right before Christmas break. We were, however, able to check off every item on our to-do list due to the fact that we had been working ahead of schedule for weeks. Luckily, we experienced no delay and could continue fresh and healthy after the break. Although we still had a fair amount of work to finish for the report and guide, we were confident we would finish in time by continuing to work how we did in the first couple of weeks. We did run a little bit behind on finishing the code because we hadn't taken into account the fact that user tests would invoke more changes to the Notebook. The user testing was changed to be an iterative process, but this led to having to process survey results at the last moment. Combined with creating the website, this was quite a lot more work than our planning indicated. Our team was well capable of dividing work among team members efficiently, and by working in dynamic pairs, we made sure everyone was kept up to date about every part of the project. We used Git to share code, Visual Studio Code to code everything and see each other's work, Liveshare to collaborate, and distributed the work during our team meetings. While some team members spent time on the code, others documented, made reflections or collected data. The team worked great together without any arguments. We also did some team bonding activities to strengthen our overall relationship with each other. The only instance of 'conflict' we ever experienced was (very) minor disagreement about our vision for the project. Suffice it to say we can all be proud of the efforts we took as a team to get to the final product we delivered here.

4.2 Recommendations

The product was created to be easily adaptable and act as a framework for further expansion. The *won'ts* and *coulds* that were scoped out are the first things that come to mind as recommendations for further research. This product could be integrated with the IoT cloud, which would allow designers to gather data more easily. The user interface could also be designed to be more user-friendly, for example, by using an API. This way, designers without coding experience won't have to learn how to open and use Notebooks. From the user tests, it became clear that it would be useful to add a page with common errors and how to solve them or even add a support functionality where users can directly contact developers. Another addition to the user evaluations would be creating another Notebook without instructions and taking surveys to investigate the effect of adding instructions.

As the output is completely interchangeable, any desired information about the processed data can be requested by writing another function. For this reason, a recommendation could be to analyse what type of information may be useful to a designer to personalise our product further for a designer's use case. This recommendation is ethically relevant. It's about adding functionalities to increase the inclusivity of the product and is explained in more detail in our third ethical reflection (Appendix G). Designers should be aware of the ethics revolving around data collection - consent, representability, and privacy need to be taken into account. In general, ethics should always be considered when working with data like this. Within future projects on which this product is built, any new features should, therefore, always be implemented with ethics in mind.

Literature

Andreas Bulling, Ulf Blanke, and Bernt Schiele. 2014. A tutorial on human activity recognition using body-worn inertial sensors. ACM Comput. Surv. 46, 3, Article 33 (January 2014), 33 pages. https://doi.org/10.1145/2499621

Bogers, S., Frens, J., van Kollenburg, J., Deckers, E., & Hummels, C. (2016). Connected baby bottle: A design case study towards a framework for data-enabled design. In Proceedings of the 2016 DIS Conference on Designing Interactive Systems (pp. 301-311). New York, NY: ACM. https://doi.org/10.1145/2901790.2901855

G., David. (2022, April 1). Getting started with GoPro Telemetry to parse GPMD. Trek View. Retrieved January 23, 2023, from https://www.trekview.org/blog/2022/gopro-telemetry-exporter-getting-started/

Gorkovenko, K., Burnett, D. J., Thorp, J. K., Richards, D. & Murray-Rust, D. (2020). Exploring The Future of Data-Driven Product Design. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3313831.3376560

Gorkovenko, K., Burnett, D., Thorp, J., Richards, D., & Murray-Rust, D. (2019). Supporting Real-Time Contextual Inquiry Through Sensor Data.

Gorkovenko, K., Burnett, D.J., Thorp, J.K., Richards, D., Murray-Rust, D. (2020). Exploring the future of data-driven product design. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (1-14). New York, NY: ACM.

Gupta, N., Gupta, S. K., Pathak, R. K., Jain, V., Rashidi, P., & Suri, J. S. (2022). Human activity recognition in artificial intelligence framework: a narrative review. *Artificial Intelligence Review*, *55*(6), 4755–4808. https://doi.org/10.1007/s10462-021-10116-x

Gupta, S. (2021). Deep learning based human activity recognition (HAR) using wearable sensor data. *International Journal of Information Management Data Insights*, *1*(2), 100046. https://doi.org/10.1016/j.jjimei.2021.100046

Human activity recognition (HAR) using machine learning. (z.d.). Neural Designer is a registered trademark of Artificial Intelligence Techniques, SL https://www.neuraldesigner.com/solutions/activity-recognition

Jagtap, S., & Duong, L. N. K. (2019). Improving the new product development using big data: a case study of a food company. *British Food Journal*, *121*(11), 2835–2848. https://doi.org/10.1108/bfj-02-2019-0097

JuanIrache, J. (2022, February 3). gpmf-extract. GitHub. Retrieved January 23, 2023, from https://github.com/JuanIrache/gpmf-extract/tree/master/samples

Kaptein, M.C., Nass, C., Markopoulos, P.. 2010. Powerful and consistent analysis of likert-type rating scales. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10). Association for Computing Machinery, New York, NY, USA*. 2391–2394. https://doi.org/10.1145/1753326

Lavanya Devi, G., & Vizianandа Row, S. (2017). A Random Forest based Classification Model for Human Activity Recognition. *International Journal of Advanced Scientific Technologies ,Engineering and Management Sciences*, *3*(1), 2454-356X. http://www.ijastems.org/wp-

content/uploads/2017/03/v3.si1_.59.A-Random-Forest-based-Classification-Model-for-Human-Activity-Recognition.pdf

Nurwulan, N. R., & Selamaj, G. (2020). Random Forest for Human Daily Activity Recognition. *Journal of Physics: Conference Series*, *1655*(1), 012087. https://doi.org/10.1088/1742-6596/1655/1/012087

Pryke, B. (2022, 4 oktober). *How to Use Jupyter Notebook: A Beginner's Tutorial*. Dataquest. https://www.dataquest.io/blog/jupyter-notebook-tutorial/

Rainock, M., Everett, D., Pack, A., Dahlin, E. C. & Mattson, C. A. (2018). The social impacts of products: a review. *Impact Assessment and Project Appraisal*, *36*(3), 230–241. https://doi.org/10.1080/14615517.2018.1445176

Talari, S. (2022, November 1). *Random Forest vs Decision Tree: Key Differences*. KDnuggets. Retrieved January 14, 2023, from https://www.kdnuggets.com/2022/02/random-forest-decision-tree-key-differences.html

Wang, Jindong, et al. "Deep learning for sensor-based activity recognition: A survey." Pattern recognition letters 119 (2019): 3-11

Yuan, W., Han, Y., Guan, D., Lee, S., & Lee, Y. K. (2011). Initial training data selection for active learning. *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*. https://doi.org/10.1145/1968613.1968619

Appendix D: Performance graphs

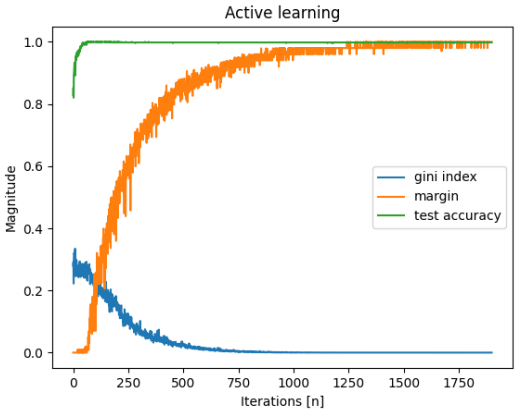


Figure D.1: *All features big dataset*

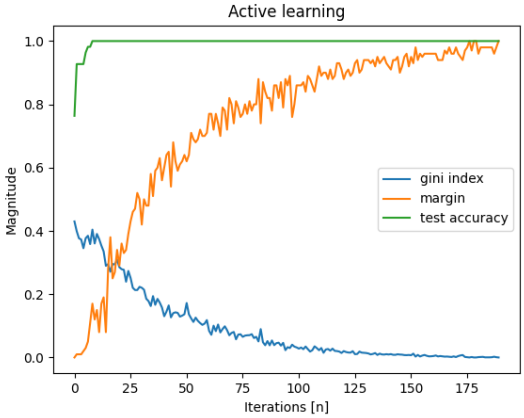


Figure D.2: *All features small dataset*

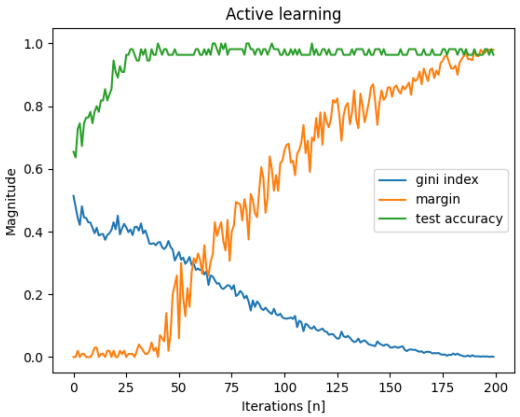


Figure D.3: *Frequency features small dataset*

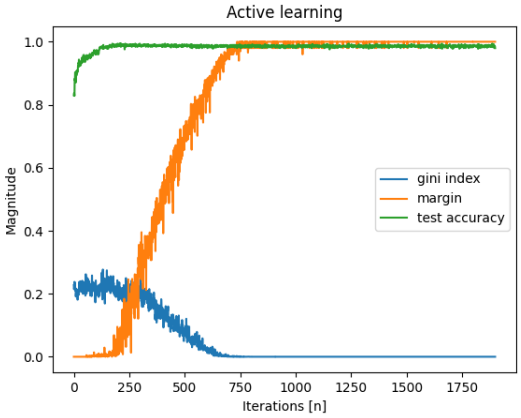


Figure D.4: *Frequency features big dataset*

Appendix E: Active Learning code

```python
def find_most_ambiguous_id(self) -> tuple[int, int | Any, Any]:
    """Finds the most ambiguous sample. The unlabeled sample with the greatest
        difference between most and second most probably classes is the most ambiguous.
        Returns only the id of this sample"""
    try:
        # Fit the model with the datapoints that we have currently labeled.
        self.model.fit(self.preds[:, 3:], self.preds[:, 1])
        # Use this model to get probabilities of datapoints belonging to a certain class.
        sorted_preds = np.sort(self.model.predict_proba(self.unpreds[:, 3:]), axis=1)
        # Basses for the lowest margins
        lowest_margin = 2
        lowest_margin_sample_id: int = 0
        # Append an empty list for the results of this iteration
        self.gini_margin_acc.append([0., 0., 0.])
        # Make a list of the unlabeled ids and sort it
        unlbld = list(self.unlabeled_ids)
        unlbld.sort()
        # Iterate for the length of datapoints that you have not yet labeled
        for i in range(sorted_preds.shape[0]):
            # Subtract from the most certain class the secon to most certain class
            margin = sorted_preds[i, -1] - sorted_preds[i, -2]
            # Is it the lowest?
            if margin < lowest_margin:
                lowest_margin_sample_id = self.unpreds[i, 0]
                lowest_margin = margin
            # Add the gini of the datapoint to gini of this iteration
            self.gini_margin_acc[-1][0] += self.gini_impurity_index(list(sorted_preds[i, :]))
        # Make it an average and add the lowest margin
        self.gini_margin_acc[-1][0] /= len(unlbld)
        self.gini_margin_acc[-1][1] = lowest_margin

        les_probs = self.model.predict_proba(
            self.unpreds[np.where(self.unpreds[:, 0] == lowest_margin_sample_id)[0], 3:]).tolist()[0]

        # Add the accuracy, this is only for a plot
        # self.gini_margin_acc[-1][2] = accuracy_score(self.model.predict(self.X_test[:, 3:]), self.y_test)
        return lowest_margin_sample_id, lowest_margin, les_probs
    # Exception mostly for testing
    except ValueError:
        raise ValueError(self.preds)
```

Figure E.1: *find_most_ambiguous_id(self) in active_learning.py*

```python
def set_starting_points(self, n_samples: int) -> None:
    """Generates training set by selecting random starting points, labeling them, and checking if there's an
    instance of every activity"""

    # Keep track of what activities we have labeled already
    seen_activities = []  # list of strings
    # Amount of datapoints that we randomly sample
    range_var = n_samples * len(self.labels)
    # Generate random points
    for i in range(range_var):
        # Pick a random point from X_pool
        while True:
            # Set a random id that is in the X_pool and has not yet been labeled
            random_id = random.randint(0, self.datapd.shape[0])
            if random_id not in self.labeled_ids and random_id in self.X_pool['ID']:
                break
        # Give the timestamp to the identification module but for testing I have automated it
        # got_labeled = self.identify(self.datapd.iloc[random_id]['time'])
        got_labeled = self.identify(random_id)  # for testing
        if got_labeled == 'x':
            print(np.where(self.datapd.iloc[:, 0] == random_id))
            self.datapd.drop(random_id, 0)
            self.X_pool.drop(random_id, 0)
        # If this label was not accounted for we add it to the set of labels
        else:
            if got_labeled not in self.labels:
                self.labels.append(got_labeled)
            seen_activities.append(got_labeled)
            # Add the ID to the list
            self.labeled_ids.append(random_id)
    # Keep adding points until every activity is in the training set (found a sample of all the labels that we expected)
    # Randomized phase is done
    for i in range(len(self.labeled_ids)):
        print(np.where(self.labeled_ids[i]), self.labeled_ids[i])
        self.X_pool.at[self.labeled_ids[i], 'label'] = seen_activities[i]
```

Figure E.2: *randomized sampling function set_starting_points(self, n_samples) in active_learning.py*

```python
def clustered_starting_points(self, n_samples: int) -> None:
    # Amount of clusters that we expect
    n_clusters = len(self.labels)
    # Determine the means
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(self.unpreds[:, 3:])
    cert_indices = []
    predictions = np.array(kmeans.predict(self.unpreds[:, 3:]))
    X = self.unpreds.copy()
    X[:, 1] = predictions
    for label in range(n_clusters):
        # Select samples with label
        x = X[np.where(X[:, 1] == label)[0], :]
        for _ in range(n_samples):  # For now
            # Transform
            total_dists = np.sum(kmeans.transform(x[:, 3:]) ** 2, axis=1)
            # Add certain samples
            cert_indices.append(x[np.argmin(total_dists), 0])
            x = np.delete(x, np.argmin(total_dists), 0)
    for e in cert_indices:
        got_labeled = self.identify(e)  # for testing
        self.labeled_ids.append(e)
        if got_labeled == 'x':
            self.remove_row(e)
        else:
            line = self.X_pool.iloc[np.where(self.X_pool.iloc[:, 0] == e)[0][0], :].copy()
            line.at['label'] = got_labeled
            line = np.array(line).reshape(1, -1)
            self.preds = np.append(self.preds, line, axis=0)
            self.unpreds = np.delete(self.unpreds, np.where(self.unpreds[:, 0] == e), 0)
```

Figure E.3: *clustered_starting_points(self, n_samples) in active_learning.py*

Appendix F: Requirements (MoSCoW)

M.1 Must support the use of accelerometer and gyroscope by correctly preprocessing the sensor data by week 4
M.2 We must have collected and preprocessed at least 30 minutes of simple sensor data before week 5
M.3 Must create a product that can distinguish between at least 2 different activities by week 6
M.4 Must create a product that is able to detect a non-classified activity by week 7
M.5 Must find the best performing ML techniques according to sensor data, and integrate it in our product before Christmas break
M.6 Must have a classifier that performs better on unseen data than our dummy classifier by week 7
M.7 Must include a guide with step by step instructions on how to use our system in the final product by week 9
S.1 Should write documentation and use code annotations to make it possible to adapt and expand our product
S.2 Should use dimension reduction to visualise behavior in plots in order for the Designer to examine the possibility of any new clusters
S.3 75% of a test panel consisting of IDE students should rate our product a 6.5 or higher
S.4 Should classify activities in at least 3 different contexts
C.1 Could implement a functionality to upload sensor data to the IoT cloud and use data from the IoT cloud
C.2 Could add a functionality of automatically updating the models using data from the real world, using an unsupervised learning technique like clustering
C.3 Could have a form of novelty detection to give designers insight into unexpected behavior and new clusters
W.1 Won't create a model that is able to process data types other than accelerometer and gyroscope
W.2 Won't design a complete system and interface that's fully integrated in the IoT cloud

Appendix H: Novelty detection code

```python
def detect(self, contamination: int = 0.1) -> list[int]:
    """Function that detects anomalies in the data using LocalOutlierFactor

    Args:
        contamination (int): The percentage of the dataset that is considered an outlier. Defaults to 0.1.

    Returns:
        list[int]: A list of the ids of the novelties
    """

    # Choosing the model LocalOutlierFactor
    clf = LocalOutlierFactor(n_neighbors=20, novelty=False, contamination=contamination, n_jobs=int(cpu_count()*3/4))
    # fit and predict the model
    prediction = clf.fit_predict(self.datapd.iloc[:, 3:])
    # Counting the outliers, which are represented with the value -1
    count = 0
    for value in prediction:
        if value == -1:
            count += 1
    # Saving a list of the outliers ids
    ids = np.where(prediction == -1)[0]

    time_video = []
    # print(self.datapd)
    for id in ids:
        with open(self.data_file) as f:
            f.readline()
            for _ in range(id):
                f.readline()
            splitted = f.readline().strip().split(',')
            i = int(splitted[0])
            time = float(splitted[2])
        with open(self.processed_data_files) as f:
                for line in f:
                    splitted = line.strip().split(',')
                    if int(splitted[1]) <= i <= int(splitted[2]):
                        video = splitted[3]
                        time_video.append([time, video])
                        break

    return time_video
```

Figure H.1: We detect the novelties with this function.

```python
def play_novelties(self, time_video: list[list[float, str]], window_size: float) -> None:
    """Function to display the video in the output cell. The video starts automatically at the timestamp,
    plays for window_size seconds and then goes back to the timestamp to loop.

    Args:
        video_file (str): relative file-location to the video file.
        timestamp (float): starting point of the window, seen from the start of the video in seconds.
        window_size (float): length of the window in seconds.
    """

    video_offset = {}
    with open(self.processed_data_files) as f:
        for line in f:
            split = line.strip().split(',')
            video_offset[split[3]] = float(split[4])

    for time_vid in time_video:
        time_vid[0] += video_offset[time_vid]

    # Function to display HTML code
    display(HTML(f'''
            <head>
                <script type="text/javascript">
                let id = 0;
                const time_video = {time_video};
                let time = time_video[0][0];
                const ws = {window_size};
```

Figure H.2: Code that will display the novelties by means of HTML.

Appendix I: PCA code

```python
    def determine_pca(self):
        """Calculates and saves the pca of the data in self.pca
        """
        pca = PCA(n_components=2, svd_solver='auto')
        self.pca = np.array(pca.fit_transform(self.datapd.iloc[:, 3:]))
        self.pca = np.append(np.array([[i for i in range(len(self.datapd))]]).reshape(-1, 1), self.pca, axis=1)
```

Figure I.1: Creating the PCA of the dataset.

```python

    def print_prediction_point(self, current_id: int):
        """Creates a file with a plot of the data and the current prediction point

        Args:
            current_id (int): ID of the current prediction point
        """
        plt.clf()
        plt.scatter(self.pca[:, 1], self.pca[:, 2], c='grey')
        plt.scatter(self.pca[current_id, 1], self.pca[current_id, 2], c='red', marker='x', label='current', s=150)
        for label in self.labels:
            # Pandas made me do it. Fuck pandas
            # lst = list(self.datapd.loc[self.datapd['label'] == label].iloc[:, 0])
            # print(np.where(self.preds[:, 1] == label))
            lst = self.preds[np.where(self.preds[:, 1] == label)[0], :]
            lst = lst[:, 0].tolist()
            # print(lst)
            temp_pca = []
            for e in self.pca:
                # print(int(e[0]), type(e.tolist()), int(e[0]) in lst)
                if int(e[0]) in lst:
                    print(e)
                    temp_pca.append(e[1:].tolist())
            # print(temp_pca, '\n')

            x = []
            y = []
            for i in range(len(temp_pca)):
                x.append(temp_pca[i][0])
                y.append(temp_pca[i][1])
            plt.scatter(x, y, label=label)
        plt.legend()
        # plt.savefig(f'Plots/plot_to_label.png')
        plt.savefig(f'Plots/plot_to_label_{self.html_id}.png')
        # self.html_id += 1

```

Figure I.2: Plotting the PCA, color code the annotated datapoints and mark the current datapoint that needs to be annotated.

Appendix K: Validation approach

	Approach	*Timeframe*
M.1	Test preprocessing pipeline using PyTest	After coding the preprocessing steps, and after further changes are made
M.2	Compute and sum the intervals between activities	It is known how long it took to collect data, only a computation needs to be used to validate this
M.3	The model can distinguish activities if the predicted labels for different activities are also different	This will be checked when the code has been finished, and afterwards keep being checked regularly
M.4	We feed our trained model data that we know has an unknown activity for the model	This must be done after model selection, and as a final check
M.5	Pick a relevant performance measure and make sure the model doesn't overfit/underfit	Start looking at performance during model selection, and keep validating it during the tuning and integrating process
M.6	Compute a dummy classifier, then compare. Also, collect or set aside some validating data	After model selection unseen data will be put in and re-trained and tuned if it performs insufficiently
M.7	We create a guide that explains the use of the product	This guide is necessary at the end of the project
S.1	If we have documentation that explains the interface and functions, and we have annotations in the code we can conclude that this requirement is met	Annotations are written while coding; one team member will check if they are present, understandable and correct in the week of handing in the project
S.2	Plots can be examined in either a 2D or a 3D plot; it should be possible to visually identify clusters	Added after exploring novelty detection, which will happen around week 7/8
S.3	Design survey for HCI research and find a representative panel	Start finding test subjects and think about research setting and questions in the first phase, take surveys in final phase
S.4	We would likely have 3 different model architectures, based on their optimal performance in each specific context. An explanation must be present to explain their differences and generalizability	First, repeat the data collection and model selection process 3 times. After having selected and implemented working models for activities from 3 different prototypes, they will be validated in the final phase
C.1	Output should be reasonable when IoT data is put into our models	This will only be done after implementing working models in 3 different contexts. Connection to the IoT cloud has a lower priority than

		previous requirements
C.2	If this feature is present, we validate model performance on an appropriate evaluation method	Only if time permits it. It is a complex feature that needs most other requirements to be met
C.3	This provides the Designer with additional information about how they could add new labels to increase performance	If time permits it, this can be done after validating that the chosen model meets requirement M.4

Appendix M: JavaScript code

This the JavaScript code used to extract the telemetry data from the GoPro video

```
const gpmfExtract = require('gpmf-extract');
const goproTelemetry = require(`gopro-telemetry`);
const fs = require('fs');

const file = fs.readFileSync('D:/DCIM/100GOPRO/GL020034_LRV.MP4');



gpmfExtract(file)
  .then(extracted => {
    goproTelemetry(extracted, {}, telemetry => {
      fs.writeFileSync('Timo_fietsen_GH020034.json', JSON.stringify(telemetry));
      console.log('Telemetry saved as JSON');
    });
  })
  .catch(error => console.error(error));
```

Figure M.1: JavaScript code to extract the telemetry data